Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 19 September 2026

B. Ryan
J. Moxey
T. Meagher
Tempo Labs
J. Weinstein
S. Kaliski
Stripe

18 March 2026

The "Payment" HTTP Authentication Scheme
draft-ryan-httpauth-payment-01

Abstract

This document defines the "Payment" HTTP authentication scheme, enabling HTTP resources to require a payment challenge to be fulfilled before access. The scheme extends HTTP Authentication, using the HTTP 402 "Payment Required" status code.

The protocol is payment-method agnostic, supporting any payment network or currency through registered payment method identifiers. Specific payment methods are defined in separate payment method specifications.

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 September 2026.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Relationship to Payment Method Specifications 4
2. Requirements Language 4
3. Terminology 4
4. Protocol Overview 4
4.1. Request Flow 4
4.2. Status Codes 5
4.3. Relationship to 401 Unauthorized 6
4.4. Usage of 402 Payment Required 7
4.4.1. When to Return 402 7
4.4.2. When NOT to Return 402 7
4.4.3. Interaction with Other Authentication Schemes 7
5. The Payment Authentication Scheme 8
5.1. Challenge (WWW-Authenticate) 8
5.1.1. Required Parameters 8
5.1.2. Optional Parameters 9
5.1.3. Request Body Digest Binding 11
5.2. Credentials (Authorization) 12
5.2.1. Example Credential 13
5.3. Payment-Receipt Header 14
5.3.1. Receipt Status Semantics 14
6. Payment Methods 15
6.1. Method Identifier Format 15
6.2. Method Registry 15
7. Payment Intents 15
7.1. Intent Identifiers 15
7.2. Intent Specifications 15
7.3. Intent Negotiation 16
8. Error Handling 16
8.1. Error Response Format 16
8.2. Error Codes 16
8.3. Retry Behavior 17
9. Extensibility 17
9.1. Payment Method Specifications 17
9.2. Versioning 17
9.2.1. Core Protocol 17
9.2.2. Payment Methods 18

9.2.3. Payment Intents 18
9.3. Custom Parameters 18
9.4. Size Considerations 18
10. Internationalization Considerations 18
10.1. Character Encoding 18
10.2. Human-Readable Text 18
11. Security Considerations 19
11.1. Threat Model 19
11.2. Transport Security 19
11.2.1. Credential Handling 19
11.3. Replay Protection 19
11.4. Idempotency and Side Effects 20
11.5. Concurrent Request Handling 20
11.6. Amount Verification 20
11.7. Privacy 20
11.8. Credential Storage 21
11.9. Intermediary Handling of 402 21
11.10. Caching 21
11.11. Cross-Origin Considerations 21
11.12. Denial of Service 21
12. IANA Considerations 22
12.1. Authentication Scheme Registration 22
12.2. Header Field Registration 22
12.3. Payment Method Registry 22
12.4. Payment Intent Registry 22
13. References 23
13.1. Normative References 23
13.2. Informative References 24
Appendix A. ABNF Collected 24
Appendix B. Examples 25
B.1. One-Time Charge 25
B.2. Signed Authorization 27
B.3. Multiple Payment Options 28
B.4. Failed Payment Verification 28
Appendix C. Acknowledgements 29
Authors' Addresses 29

1. Introduction

HTTP 402 "Payment Required" was reserved in HTTP/one-point-one [RFC9110] but never standardized for common use. This specification defines the "Payment" authentication scheme that gives 402 its semantics, enabling resources to require a payment challenge to be fulfilled before access.

1.1. Relationship to Payment Method Specifications

This specification defines the abstract protocol framework. Concrete payment methods are defined in payment method specifications that:

* Register a payment method identifier
* Define the request schema for that method
* Define the payload schema for that method
* Specify 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

Payment Challenge A WWW-Authenticate header with scheme "Payment" indicating the payment requirements for accessing a resource.

Payment Credential An Authorization header with scheme "Payment" containing payment authorization data.

Payment Method A mechanism for transferring value, identified by a registered identifier.

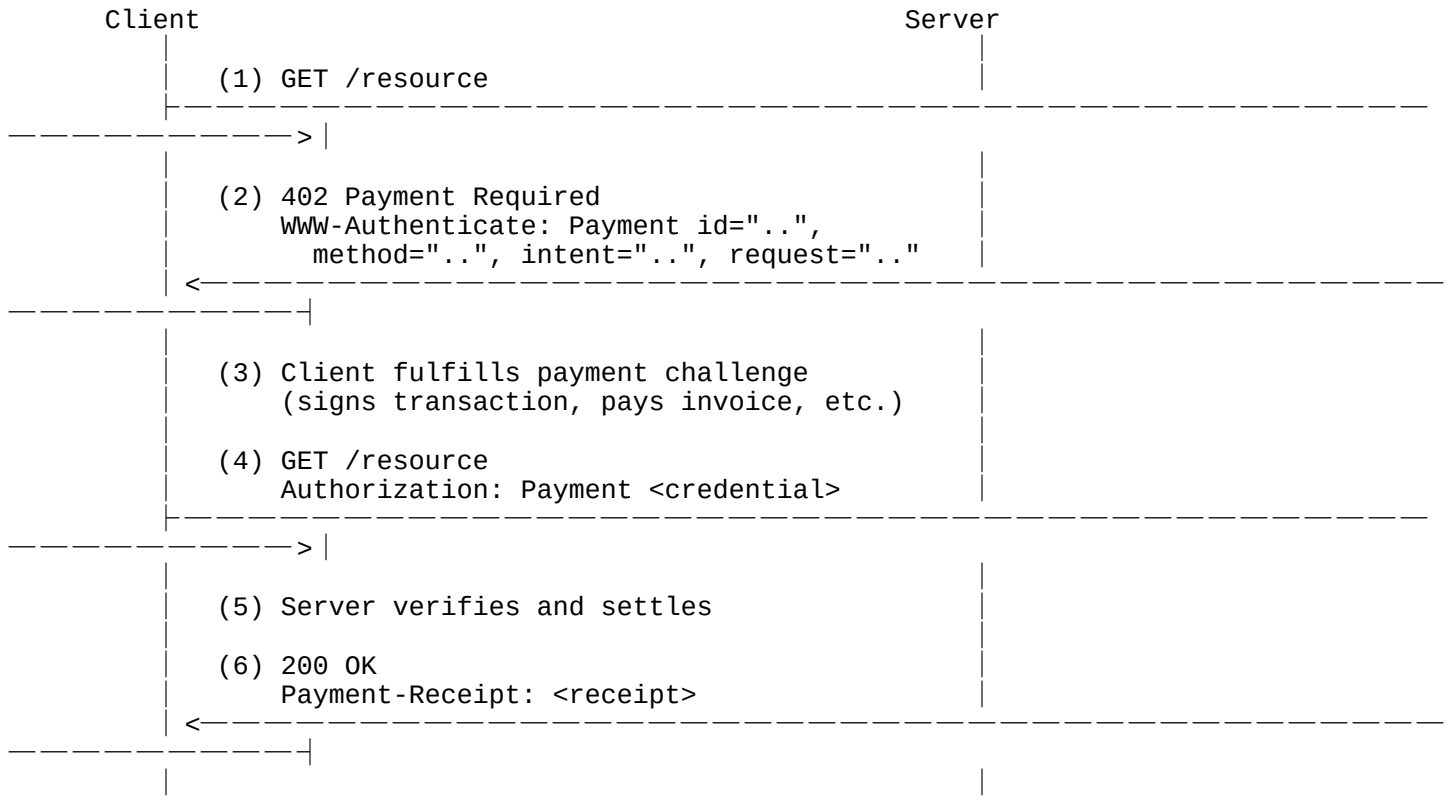
Payment Intent The type of payment request, identified by a registered value in the IANA "HTTP Payment Intents" registry. Intents are defined by separate intent specifications.

Request Method-specific data in the challenge enabling payment completion. Encoded as base64url JSON in the request parameter.

Payload Method-specific data in the credential proving payment.

4. Protocol Overview

4.1. Request Flow



4.2. Status Codes

The following table defines how servers MUST respond to payment-related conditions.

Condition	Status	Response
Resource requires payment, no credential provided	402	Fresh challenge in WWW-Authenticate
Malformed credential (invalid base64url, bad JSON)	402	Fresh challenge + malformed-credential problem
Unknown, expired, or already-used challenge id	402	Fresh challenge + invalid-challenge problem
Payment proof invalid or verification failed	402	Fresh challenge + verification-failed problem
Payment verified, access granted	200	Resource + optional Payment-Receipt
Payment verified, but policy denies access	403	No challenge (payment was valid)

Table 1

Servers MUST return 402 with a WWW-Authenticate: Payment header when payment is required or when a payment credential fails validation (see Section 4.4 for details).

Error details are provided in the response body using Problem Details [RFC9457] rather than in the WWW-Authenticate header parameters.

4.3. Relationship to 401 Unauthorized

This specification uses 402 (Payment Required) consistently for all payment-related challenges, including failed credential validation. This diverges from the traditional 401 pattern used by other HTTP authentication schemes. The distinction is intentional:

* *402* indicates a payment barrier (initial challenge or retry needed)

* *401* is reserved for authentication failures unrelated to payment

* *403* indicates the payment succeeded but access is denied by policy

This design ensures clients can distinguish payment requirements from other authentication schemes that use 401.

4.4. Usage of 402 Payment Required

4.4.1. When to Return 402

Servers SHOULD return 402 when:

* The resource requires payment as a precondition for access
* The server can provide a Payment challenge that the client may fulfill
* Payment is the primary barrier to access (not authentication or authorization)

Servers MAY return 402 when:

* Offering optional paid features or premium content
* Indicating that a previously-paid resource requires additional payment
* The payment requirement applies to a subset of request methods

4.4.2. When NOT to Return 402

Servers SHOULD NOT return 402 when:

* The client lacks authentication credentials (use 401)
* The client is authenticated but lacks authorization (use 403)
* The resource does not exist (use 404)
* No Payment challenge can be constructed for the request

Servers MUST NOT return 402 without including a WWW-Authenticate header containing at least one Payment challenge.

4.4.3. Interaction with Other Authentication Schemes

When a resource requires both authentication and payment, servers SHOULD:

1. First verify authentication credentials

2. Return 401 if authentication fails

3. Return 402 with a Payment challenge only after successful authentication

This ordering prevents information leakage about payment requirements to unauthenticated clients.

5. The Payment Authentication Scheme

5.1. Challenge (WWW-Authenticate)

The Payment challenge is sent in the WWW-Authenticate header per [RFC9110]. The challenge uses the auth-param syntax defined in Section 11 of [RFC9110]:

```
challenge       = "Payment" [ 1*SP auth-params ]
auth-params     = auth-param *( OWS "," OWS auth-param )
auth-param      = token BWS "=" BWS ( token / quoted-string )
```

5.1.1. Required Parameters

id: Unique challenge identifier. Servers MUST bind this value to the challenge parameters (Section 5.1.3) to enable verification. Clients MUST include this value unchanged in the credential.

realm: Protection space identifier per [RFC9110]. Servers MUST include this parameter to define the scope of the payment requirement.

method: Payment method identifier (Section 6). MUST be a lowercase ASCII string.

intent: Payment intent type (Section 7). The value MUST be a registered entry in the IANA "HTTP Payment Intents" registry.

request: Base64url-encoded [RFC4648] JSON [RFC8259] containing payment-method-specific data needed to complete payment. Structure is defined by the payment method specification. Padding characters ("=") MUST NOT be included. The JSON MUST be serialized using JSON Canonicalization Scheme (JCS) [RFC8785] to ensure deterministic encoding across implementations. This is critical for challenge binding (Section 5.1.2.1): since the HMAC input includes the base64url-encoded request as it appears on the wire, different JSON serialization orders would produce different HMAC values, breaking cross-implementation interoperability.

5.1.2. Optional Parameters

digest: Content digest of the request body, formatted per [RFC9530]. Servers SHOULD include this parameter when the payment challenge applies to a request with a body (e.g., POST, PUT, PATCH). When present, clients MUST submit the credential with a request body whose digest matches this value. See Section 5.1.3 for body binding requirements.

expires: Timestamp indicating when this challenge expires, formatted as an [RFC3339] date-time string (e.g., "2025-01-15T12:00:00Z"). Servers SHOULD include this parameter. Clients MUST NOT submit credentials for expired challenges.

description: Human-readable description of the resource or payment purpose. This parameter is for display purposes only and MUST NOT be relied upon for payment verification (see Section 11.6).

opaque: Base64url-encoded [RFC4648] JSON [RFC8259] containing server-defined correlation data (e.g., a payment processor intent identifier). The value MUST be a JSON object whose values are strings (a flat string-to-string map). Clients MUST return this parameter unchanged in the credential and MUST NOT modify it. The JSON MUST be serialized using JSON Canonicalization Scheme (JCS) [RFC8785] before base64url encoding. Servers SHOULD include opaque in the challenge binding (Section 5.1.2.1) to ensure tamper protection.

Unknown parameters MUST be ignored by clients.

5.1.2.1. Challenge Binding

Servers SHOULD bind the challenge id to the challenge parameters (Section 5.1.1 and Section 5.1.2) to prevent request integrity attacks where a client could sign or submit a payment different from what the server intended. Servers MUST verify that credentials present an id matching the expected binding.

The binding mechanism is implementation-defined. Servers MAY use stateful storage (e.g., database lookup) or stateless verification (e.g., HMAC, authenticated encryption) to validate the binding.

5.1.2.1.1. Recommended: HMAC-SHA256 Binding

Servers using HMAC-SHA256 for stateless challenge binding SHOULD compute the challenge id as follows:

The HMAC input is constructed from exactly seven fixed positional slots. Required fields supply their string value; optional fields use an empty string ("") when absent. The slots are:

Slot	Field	Value
0	realm	Required. String value.
1	method	Required. String value.
2	intent	Required. String value.
3	request	Required. JCS-serialized per [RFC8785], then base64url-encoded.
4	expires	Optional. String value if present; empty string if absent.
5	digest	Optional. String value if present; empty string if absent.
6	opaque	Optional. JCS-serialized per [RFC8785], then base64url-encoded if present; empty string if absent.

Table 2

The computation proceeds as follows:

1. Populate all seven slots as described above.

2. Join all seven slots with the pipe character (|) as delimiter. Every slot is always present in the joined string; absent optional fields appear as empty segments (e.g., ...|expires||opaque_b64url when digest is absent).

3. Compute HMAC-SHA256 over the resulting string using a server secret.

4. Encode the HMAC output as base64url without padding ([RFC4648] Section 5).

```
input = "|".join([
    realm,
    method,
    intent,
    request_b64url,
    expires or "",
    digest or "",
    opaque_b64url or "",
])
id = base64url(HMAC-SHA256(server_secret, input))
```

Optional fields use fixed positional slots with empty strings when absent, rather than being omitted. This avoids ambiguity between combinations of optional fields — for example, (expires set, no digest) and (no expires, digest set) produce distinct inputs — and ensures that adding a new optional slot in a future revision does not change the HMAC for challenges that omit it.

5.1.2.2. Example Challenge

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="x7Tg2pLqR9mKvNwY3hBcZa",
    realm="api.example.com",
    method="example",
    intent="charge",
    expires="2025-01-15T12:05:00Z",
    request="eyJhbW91bnQiOiIxMDAwIiwiY3VycmVuY3kiOiJVU0QiLCJyZWNpcGllbnQiOiJhY2N0XzEyM
yJ9"
```

Decoded request example:

```
{
  "amount": "1000",
  "currency": "usd",
  "recipient": "acct_123"
}
```

5.1.3. Request Body Digest Binding

Servers SHOULD include the digest parameter when issuing challenges for requests with bodies. The digest value is computed per [RFC9530]:

```
WWW-Authenticate: Payment id="...",
    realm="api.example.com",
    method="example",
    intent="charge",
    digest="sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:",
    expires="2025-01-15T12:05:00Z",
    request="..."
```

When verifying a credential with a digest parameter, servers MUST:

1. Compute the digest of the current request body per [RFC9530]

2. Compare it with the digest value from the challenge

3. Reject the credential if the digests do not match

5.2. Credentials (Authorization)

The Payment credential is sent in the Authorization header using base64url encoding without padding per [RFC4648] Section 5:

```
credentials     = "Payment" 1*SP base64url-nopad
base64url-nopad = 1*( ALPHA / DIGIT / "-" / "_" )
```

The base64url-nopad value is a base64url-encoded JSON object (without padding) containing:

Field	Type	Required	Description
challenge	object	Yes	Echoed challenge parameters
source	string	No	Payer identifier (RECOMMENDED: DID format per [W3C-DID])
payload	object	Yes	Method-specific payment proof

Table 3

The challenge object contains the parameters from the original challenge:

Field	Type	Description
id	string	Challenge identifier
realm	string	Protection space
method	string	Payment method identifier
intent	string	Payment intent type
request	string	Base64url-encoded payment request
description	string	Human-readable payment purpose (if present in challenge)
opaque	string	Base64url-encoded server correlation data (if present in challenge)
digest	string	Content digest
expires	string	Challenge expiration timestamp

Table 4

The payload field contains the payment-method-specific data needed to complete the payment challenge. Payment method specifications define the exact structure.

5.2.1. Example Credential

```
GET /api/data HTTP/1.1
Host: api.example.com
Authorization: Payment eyJjaGFsbGVuZ2UiOnsiaWQiOiJ4N1RnMnBMcVI5bUt2TndZM2hCY1phIiwicmV
hbG0iOiJhcGkuZXhhbXBsZS5jb20iLCJtZXRob2QiOiJleGFtcGxlIiwiaW50ZW50IjoiY2hhcmdlIiwicmVxdWVz
dCI6ImV5SmhiVzkxYm5RaU9pSXhNREF3SWl3aVkzVnljbVZ1WTNraU9pSlZVMFFpTENKeVpXTnBjR2xsYm5RaU9pS
mhZMk4wWHpFeU15SjkiLCJleHBpcmVzIjoiMjAyNS0wMS0xNVQxMjowNTowMFoifSwicGF5bG9hZCI6eyJwcm9vZi
I6IjB4YWJjMTIzLi4uIn19
```

Decoded credential:

```
   {
     "challenge": {
       "id": "x7Tg2pLqR9mKvNwY3hBcZa",
       "realm": "api.example.com",
       "method": "example",
       "intent": "charge",
       "request": "eyJhbW91bnQiOiIxMDAwIiwiY3VycmVuY3kiOiJVU0QiLCJyZWNpcGllbnQiOiJhY2N0XzEyMyJ9",
       "expires": "2025-01-15T12:05:00Z"
     },
     "payload": {
       "proof": "0xabc123..."
     }
   }
```


5.3. Payment-Receipt Header

Servers SHOULD include a Payment-Receipt header on successful responses:

Payment-Receipt = base64url-nopad

The decoded JSON object contains:

Field	Type	Description
status	string	"success" — receipts are only issued on successful payment
method	string	Payment method used
timestamp	string	[RFC3339] settlement timestamp
reference	string	Method-specific reference (tx hash, invoice id, etc.)

Table 5

Payment method specifications MAY define additional fields for receipts.

5.3.1. Receipt Status Semantics

The status field MUST be "success", indicating the payment was verified and settled successfully. Receipts are only issued on successful payment responses (2xx status codes).

Servers MUST NOT return a Payment-Receipt header on error responses. Payment failures are communicated via HTTP status codes and Problem Details [RFC9457]. Servers MUST return 402 with a fresh challenge and appropriate problem type when payment verification fails.

6. Payment Methods

6.1. Method Identifier Format

Payment methods are identified by lowercase ASCII letters:

```
payment-method-id = 1*LOWERALPHA
```

Method identifiers are case-sensitive and MUST be lowercase.

6.2. Method Registry

Payment methods are registered in the HTTP Payment Methods registry (Section 12.3). Each registered method has an associated specification that defines the request and payload schemas.

7. Payment Intents

Payment intents describe the type of payment being requested.

7.1. Intent Identifiers

```
intent = 1*( ALPHA / DIGIT / "-" )
```

7.2. Intent Specifications

Payment intents are defined in separate intent specifications that:

* Define the semantic meaning of the intent
* Specify required and optional request fields
* Specify payload requirements
* Define verification and settlement semantics
* Register the intent in the Payment Intent Registry (Section 12.4)

See the Payment Intent Registry for registered intents.

7.3. Intent Negotiation

If a server supports multiple intents, it MAY issue multiple challenges:

```
WWW-Authenticate: Payment id="abc", realm="api.example.com", method="example", intent="charge", request="..."
WWW-Authenticate: Payment id="def", realm="api.example.com", method="example", intent="authorize", request="..."
```

Clients choose which challenge to respond to. Clients that do not recognize an intent SHOULD treat the challenge as unsupported.

8. Error Handling

8.1. Error Response Format

Servers SHOULD return Problem Details [RFC9457] error bodies with 402 responses:

```
{
  "type": "https://paymentauth.org/problems/payment-required",
  "title": "Payment Required",
  "status": 402,
  "detail": "Human-readable description"
}
```

The type URI SHOULD correspond to one of the problem types defined below, and the canonical base URI for problem types is https://paymentauth.org/problems/.

8.2. Error Codes

Code	HTTP	Description
payment-required	402	Resource requires payment
payment-insufficient	402	Amount too low
payment-expired	402	Challenge or authorization expired
verification-failed	402	Proof invalid
method-unsupported	400	Method not accepted
malformed-credential	402	Invalid credential format
invalid-challenge	402	Challenge ID unknown, expired, or already used

Table 6

8.3. Retry Behavior

Servers SHOULD use the Retry-After HTTP header [RFC9110] to indicate when clients may retry:

```
HTTP/1.1 402 Payment Required
Retry-After: 60
WWW-Authenticate: Payment ...
```

9. Extensibility

9.1. Payment Method Specifications

Payment method specifications MUST define:

1. *Method Identifier*: Unique lowercase string
2. *Request Schema*: JSON structure for the request parameter
3. *Payload Schema*: JSON structure for credential payloads
4. *Verification Procedure*: How servers validate proofs
5. *Settlement Procedure*: How payment is finalized
6. *Security Considerations*: Method-specific threats and mitigations

9.2. Versioning

The Payment scheme uses a layered versioning strategy:

9.2.1. Core Protocol

The Payment scheme name is the stable identifier. The core protocol does NOT carry a version on the wire, consistent with all deployed HTTP authentication schemes (Basic, Bearer, Digest). Evolution happens through adding optional parameters and fields; implementations MUST ignore unknown parameters and fields. If a future change is truly incompatible, a new scheme name (e.g., Payment2) would be registered.

9.2.2. Payment Methods

Payment method specifications MAY include a version field in their methodDetails. The absence of a version field is implicitly version 1. When a breaking change is needed, the method specification adds a version field starting at 2. Compatible changes (adding optional fields, defining defaults) do not require a version change. Methods MAY also register a new identifier for changes fundamental enough to warrant a distinct name.

9.2.3. Payment Intents

Payment intents do not carry a version. They evolve through the same compatibility rules as the core: adding optional fields with defined defaults is compatible, and breaking changes require a new intent identifier (e.g., charge-v2).

9.3. Custom Parameters

Implementations MAY define additional parameters in challenges:

* Parameters MUST use lowercase names
* Unknown parameters MUST be ignored by clients

9.4. Size Considerations

Servers SHOULD keep challenges under 8KB. Clients MUST be able to handle challenges of at least 4KB. Servers MUST be able to handle credentials of at least 4KB.

10. Internationalization Considerations

10.1. Character Encoding

All string values use UTF-8 encoding [RFC3629]:

* The request and credential payloads are JSON [RFC8259]
* Payment method identifiers are restricted to ASCII lowercase
* The realm parameter SHOULD use ASCII-only values per [RFC9110]

10.2. Human-Readable Text

The description parameter may contain localized text. Servers SHOULD use the Accept-Language request header [RFC9110] to determine the appropriate language.

11. Security Considerations

11.1. Threat Model

This specification assumes:

* Attackers can observe all network traffic

* Attackers can inject, modify, or replay messages

* Attackers may control malicious servers or clients

11.2. Transport Security

This specification REQUIRES TLS 1.2 [RFC5246] or later for all Payment authentication flows. TLS 1.3 [RFC8446] is RECOMMENDED.

Implementations MUST use TLS when transmitting Payment challenges and credentials. Payment credentials contain sensitive authorization data that could result in financial loss if intercepted.

Servers MUST NOT issue Payment challenges over unencrypted HTTP. Clients MUST NOT send Payment credentials over unencrypted HTTP. Implementations SHOULD reject Payment protocol messages received over non-TLS connections.

11.2.1. Credential Handling

Payment credentials are bearer tokens that authorize financial transactions. Servers and intermediaries MUST NOT log Payment credentials or include them in error messages, debugging output, or analytics. Credential exposure could enable replay attacks or unauthorized payments.

Implementations MUST treat Payment credentials with the same care as authentication passwords or session tokens. Credentials SHOULD be stored only in memory and cleared after use.

11.3. Replay Protection

Payment methods used with this specification MUST provide single-use proof semantics. A payment proof MUST be usable exactly once; subsequent attempts to use the same proof MUST be rejected by the payment method infrastructure.

11.4. Idempotency and Side Effects

Servers MUST NOT perform side effects (database writes, external API calls, resource creation) for requests that have not been paid. The unpaid request that triggers a 402 challenge MUST NOT modify server state beyond recording the challenge itself.

For non-idempotent methods (POST, PUT, DELETE), servers SHOULD accept an Idempotency-Key header to enable safe client retries. When a client retries a request with the same Idempotency-Key and a valid Payment credential, the server SHOULD return the same response as the original successful request without re-executing the operation.

11.5. Concurrent Request Handling

Servers MUST ensure that concurrent requests with the same Payment credential result in at most one successful payment settlement and one resource delivery. Race conditions between parallel requests could otherwise cause double-payment or double-delivery.

Implementations SHOULD use atomic operations or distributed locks when verifying and consuming Payment credentials. The credential verification and resource delivery SHOULD be performed as an atomic operation where possible.

11.6. Amount Verification

Clients MUST verify before authorizing payment:

1. Requested amount is reasonable for the resource
2. Recipient/address is expected
3. Currency/asset is as expected
4. Validity window is appropriate

Clients MUST NOT rely on the description parameter for payment verification. Malicious servers could provide a misleading description while the actual request payload requests a different amount.

11.7. Privacy

* Servers MUST NOT require user accounts for payment.
* Payment methods SHOULD support pseudonymous options where possible.

* Servers SHOULD NOT log Payment credentials in plaintext

11.8. Credential Storage

Implementations MUST treat Authorization: Payment headers and Payment-Receipt headers as sensitive data.

11.9. Intermediary Handling of 402

HTTP intermediaries (proxies, caches, CDNs) may not recognize 402 as an authentication challenge in the same way they handle 401. While this specification uses WWW-Authenticate headers with 402 responses following the same syntax as [RFC9110], intermediaries that perform special processing for 401 (such as stripping credentials or triggering authentication prompts) may not apply the same behavior to 402.

Servers SHOULD NOT rely on intermediary-specific handling of 402 responses. Clients MUST be prepared to receive 402 responses through any intermediary.

11.10. Caching

Payment challenges contain unique identifiers and time-sensitive payment data that MUST NOT be cached or reused. To prevent challenge replay and stale payment information:

Servers MUST send Cache-Control: no-store [RFC9111] with 402 responses; this ensures no shared cache reuse.

Responses containing Payment-Receipt headers MUST include Cache-Control: private to prevent shared caches from storing payment receipts.

11.11. Cross-Origin Considerations

Clients (particularly browser-based wallets) SHOULD:

* Clearly display the origin requesting payment

* Require explicit user confirmation before authorizing payments

* Not automatically respond to Payment challenges

11.12. Denial of Service

Servers SHOULD implement rate limiting on challenges issued and credential verification attempts.

12. IANA Considerations

12.1. Authentication Scheme Registration

This document registers the "Payment" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" established by [RFC9110]:

* *Authentication Scheme Name*: Payment

* *Reference*: This document, Section 5

* *Notes*: Used with HTTP 402 status code for proof-of-payment flows

12.2. Header Field Registration

This document registers the following header fields:

Field Name	Status	Reference
Payment-Receipt	permanent	This document, Section 5.3

Table 7

12.3. Payment Method Registry

This document establishes the "HTTP Payment Methods" registry. This registry uses the "Specification Required" policy defined in [RFC8126].

Registration requests must include:

* *Method Identifier*: Unique lowercase ASCII letters (a-z)

* *Description*: Brief payment-method description

* *Specification pointer*: Reference to the specification document

* *Registrant Contact*: Contact information for the registrant

12.4. Payment Intent Registry

This document establishes the "HTTP Payment Intents" registry. This registry uses the "Specification Required" policy defined in [RFC8126].

Registration requests must include:

* *Intent Identifier*: Unique lowercase ASCII string

* *Description*: Brief description of the intent semantics

* *Specification pointer*: Reference to the specification document

* *Registrant Contact*: Contact information for the registrant

The registry is initially empty. Intent specifications register their identifiers upon publication.

13. References

13.1. 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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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>.

[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>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

[RFC9530] Polli, R. and L. Pardue, "Digest Fields", RFC 9530, DOI 10.17487/RFC9530, February 2024, <https://www.rfc-editor.org/info/rfc9530>.

13.2. Informative References

[W3C-DID] W3C, "Decentralized Identifiers (DIDs) v1.0", n.d., <https://www.w3.org/TR/did-core/>.

[W3C-PMI] W3C, "Payment Method Identifiers", n.d., <https://www.w3.org/TR/payment-method-id/>.

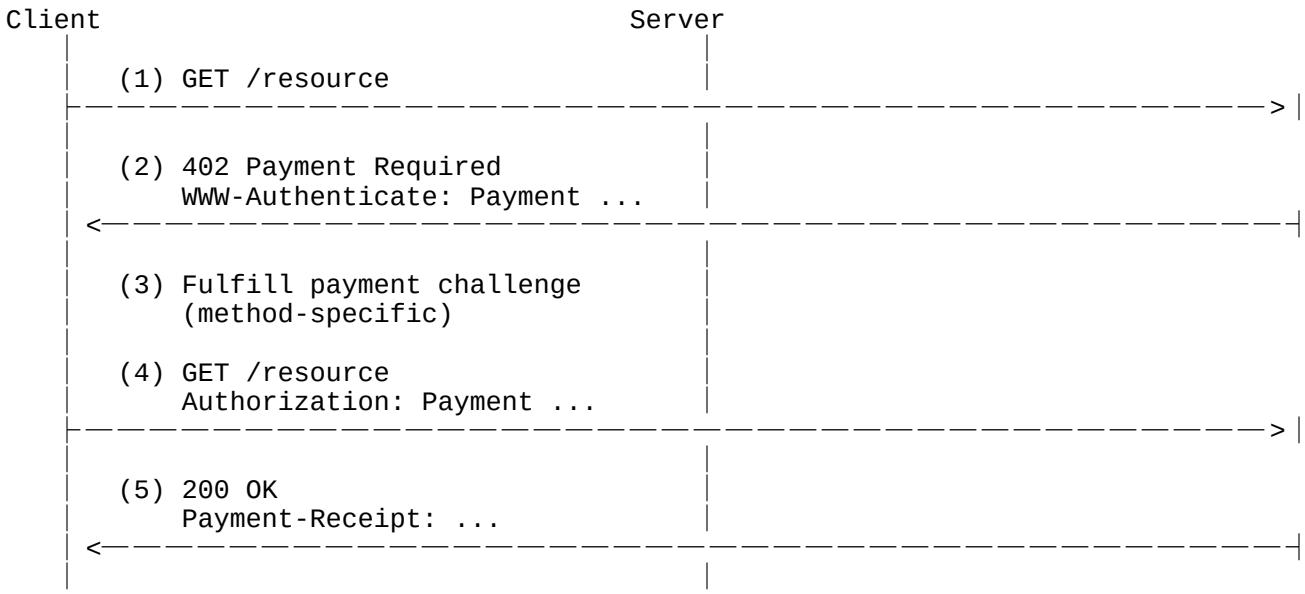
Appendix A. ABNF Collected

```
; HTTP Authentication Challenge (following RFC 7235 Section 2.1)
payment-challenge = "Payment" [ 1*SP auth-params ]
auth-params       = auth-param *( OWS "," OWS auth-param )
auth-param        = token BWS "=" BWS ( token / quoted-string )

; Required parameters: id, realm, method, intent, request
; Optional parameters: expires, digest, description, opaque

; HTTP Authorization Credentials
payment-credentials = "Payment" 1*SP base64url-nopad

; Payment-Receipt header field value
Payment-Receipt = base64url-nopad

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

; Payment method identifier (lowercase only)
payment-method-id   = 1*LOWERALPHA
LOWERALPHA          = %x61-7A  ; a-z

; Payment intent
intent-token = 1*( ALPHA / DIGIT / "-" )
```

Appendix B. Examples

B.1. One-Time Charge

A client requests a resource, receives a payment challenge, fulfills the payment, and receives the resource with a receipt.



Challenge:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
Content-Type: application/problem+json
WWW-Authenticate: Payment id="qB3wErTyU7iOpAsD9fGhJk",
    realm="api.example.com",
    method="invoice",
    intent="charge",
    expires="2025-01-15T12:05:00Z",
    request="eyJhbW91bnQiOiIxMDAwIiwiY3VycmVuY3kiOiJVU0QiLCJpbnZvaWNlIjoiaW52XzEyMzQ1I
n0"
```

```
{
  "type": "https://paymentauth.org/problems/payment-required",
  "title": "Payment Required",
  "status": 402,
  "detail": "Payment required for access.",
  "challengeId": "qB3wErTyU7iOpAsD9fGhJk"
}
```

Decoded request:

```
{
  "amount": "1000",
  "currency": "usd",
  "invoice": "inv_12345"
}
```

Credential:

```
GET /resource HTTP/1.1
Host: api.example.com
Authorization: Payment eyJpZCI6InFCM3dFclR5VTdpT3BBc0Q5ZkdoSmsiLCJwYXlsb2FkIjp7InByZWl
tYWdlIjoiMHhhYmMxMjMuLi4ifX0
```

Decoded credential:

```
{
  "challenge": {
    "id": "qB3wErTyU7iOpAsD9fGhJk",
    "realm": "api.example.com",
    "method": "invoice",
    "intent": "charge",
    "request": "eyJhbW91bnQiOiIxMDAwIiwiY3VycmVuY3kiOiJVU0QiLCJpbnZvaWNlIjoiaW52XzEyMz
Q1In0",
    "expires": "2025-01-15T12:05:00Z"
  },
  "payload": {
    "preimage": "0xabc123..."
  }
}
```

Success:

```
HTTP/1.1 200 OK
Cache-Control: private
Payment-Receipt: eyJzdGF0dXMiOiJzdWNjZXNzIiwibWV0aG9kIjoiaW52b2ljZSIsInRpbWVzdGFtcCI6I
jIwMjUtMDEtMTVUMTI6MDA6MDBaIiwicmVmZXJlbmNlIjoiaW52XzEyMzQ1In0
Content-Type: application/json

{"data": "..."}
```

B.2. Signed Authorization

A payment method using cryptographic signatures:

Challenge:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="zL4xCvBnM6kJhGfD8sAaWe",
    realm="api.example.com",
    method="signed",
    intent="charge",
    expires="2025-01-15T12:05:00Z",
    request="eyJhbW91bnQiOiI1MDAwIiwiYXNzZXQiOiJVU0QiLCJyZWNpcGllbnQiOiIweDc0MmQzNUNjN
jYzNEMwNTMyOTI1YTNiODQ0QmM5ZTc1OTVmOGZFMDAiLCJub25jZSI6IjB4MTIzNDU2Nzg5MCJ9"
```

Decoded request:

```
   {
     "amount": "5000",
     "currency": "usd",
     "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00",
     "methodDetails": {
       "nonce": "0x1234567890"
     }
   }
```

Credential:

```
   {
     "challenge": {
       "id": "zL4xCvBnM6kJhGfD8sAaWe",
       "realm": "api.example.com",
       "method": "signed",
       "intent": "charge",
       "request": "eyJhbW91bnQiOiI1MDAwIiwiYXNzZXQiOiJVU0QiLCJyZWNpcGllbnQiOiIweDc0MmQzNU
NjNjYzNEMwNTMyOTI1YTNiODQ0QmM5ZTc1OTVmOGZFMDAiLCJub25jZSI6IjB4MTIzNDU2Nzg5MCJ9",
       "expires": "2025-01-15T12:05:00Z"
     },
     "source": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK",
     "payload": {
       "signature": "0x1b2c3d4e5f..."
     }
   }
```

B.3. Multiple Payment Options

Servers MAY return multiple Payment challenges in a single 402 response, each with a different payment method or configuration:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="pT7yHnKmQ2wErXsZ5vCbNl", realm="api.example.com", method
="invoice", intent="charge", request="..."
WWW-Authenticate: Payment id="mF8uJkLpO3qRtYsA6wDcVb", realm="api.example.com", method
="signed", intent="charge", request="..."
```

When a server returns multiple challenges, clients SHOULD select one based on their capabilities and user preferences. Clients MUST send only one Authorization: Payment header in the subsequent request, corresponding to the selected challenge.

Servers receiving multiple Payment credentials in a single request SHOULD reject with 400 (Bad Request).

B.4. Failed Payment Verification

```
   HTTP/1.1 402 Payment Required
   Cache-Control: no-store
   Content-Type: application/problem+json
   WWW-Authenticate: Payment id="aB1cDeF2gHiJ3kLmN4oPqR", realm="api.example.com", method
="invoice", intent="charge", request="..."

   {
     "type": "https://paymentauth.org/problems/verification-failed",
     "title": "Payment Verification Failed",
     "status": 402,
     "detail": "Invalid payment proof."
   }
```

The server returns 402 with a fresh challenge, allowing the client to retry with a new payment credential.

Appendix C. Acknowledgements

TBD

Authors' Addresses

Brendan Ryan
Tempo Labs
Email: brendan@tempo.xyz

Jake Moxey
Tempo Labs
Email: jake@tempo.xyz

Tom Meagher
Tempo Labs
Email: tom@tempo.xyz

Jeff Weinstein
Stripe
Email: jweinstein@stripe.com

Steve Kaliski
Stripe
Email: stevekaliski@stripe.com