Workgroup: Network Working Group
Internet-Draft: draft-lightning-charge-00
Published: 18 March 2026
Intended Status: Informational
Expires: 19 September 2026

Authors: K. Zhang *Lightspark* J. Klein *Lightspark* Z. Lu *Lightspark*

Lightning Network Charge Intent for HTTP Payment Authentication

Abstract

This document defines the "charge" intent for the "lightning" payment method within the Payment HTTP Authentication Scheme [I-D.httpauth-payment]. The server issues a BOLT11 invoice as a challenge; the client pays it and proves payment by presenting the preimage as a credential.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP Payment Authentication [I-D.httpauth-payment] defines a challenge-response mechanism that gates access to resources behind micropayments. This document registers the "charge" intent for the "lightning" payment method.

The flow proceeds as follows:

```
    Client                        Server                   Lightning Network
       |                             |                             |
       |  (1) GET /resource          |                             |
       |-------------------------->  |                             |
       |                             |                             |
       |                             |  (2) Create invoice         |
       |                             |-------------------------->  |
       |                             |  (3) invoice, hash          |
       |                             |<--------------------------  |
       |                             |                             |
       |  (4) 402 Payment Required   |                             |
       |      (invoice, hash)        |                             |
       |<--------------------------  |                             |
       |                             |                             |
       |  (5) Pay invoice            |                             |
       |------------------------------------------------------>    |
       |  (6) Preimage (HTLC)        |                             |
       |<------------------------------------------------------    |
       |                             |                             |
       |  (7) GET /resource          |                             |
       |      credential: preimage   |                             |
       |-------------------------->  |                             |
       |                             |                             |
       |  (8) 200 OK (resource)      |                             |
       |<--------------------------  |                             |
       |                             |                             |
```

1.1. Relationship to the Charge Intent

This document inherits the shared request semantics of the "charge" intent from [I-D.payment-intent-charge]. It defines only the Lightning-specific `methodDetails`, `payload`, and settlement procedures for the "lightning" payment method.

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

BOLT11 Invoice A Lightning Network payment request encoded per the BOLT11 specification, containing a payment hash, amount, expiry, and optional routing hints.

Payment Hash A 32-byte SHA-256 hash of the payment preimage, encoded as a lowercase hex string. Embedded in the BOLT11 invoice and used by the server to verify payment.

Payment Preimage A 32-byte random secret whose SHA-256 hash equals the payment hash. Revealed to the payer upon successful payment settlement; serves as proof of payment.

4. Intent Identifier

The intent identifier for this specification is "charge". It **MUST** be lowercase.

5. Intent: "charge"

The "charge" intent represents a one-time payment gating access to a resource. The server generates a fresh BOLT11 invoice ([BOLT11]) per request. The client pays the invoice on the Lightning Network and presents the payment preimage as the credential. The server verifies the preimage cryptographically without contacting any external service.

6. Encoding Conventions

All JSON [RFC8259] objects carried in auth-params or HTTP headers in this specification **MUST** be serialized using the JSON Canonicalization Scheme (JCS) [RFC8785] before encoding. JCS produces a deterministic byte sequence, which is required for any digest or signature operations defined by the base spec [I-D.httpauth-payment].

The resulting bytes **MUST** then be encoded using base64url [RFC4648] Section 5 without padding characters (=). Implementations **MUST NOT** append `=` padding when encoding, and **MUST** accept input with or without padding when decoding.

This encoding convention applies to: the `request` auth-param in `WWW-Authenticate`, the credential token in `Authorization`, and the receipt token in `Payment-Receipt`.

7. Request Schema

7.1. Shared Fields

The `request` auth-param of the `WWW-Authenticate: Payment` header contains a JCS-serialized, base64url-encoded JSON object (see Section 6). The following shared fields are included in that object:

amount **REQUIRED**. The invoice amount in base units (satoshis), encoded as a decimal string (e.g., "100"). The value **MUST** be a positive integer.

currency **REQUIRED**. Identifies the unit for `amount`. **MUST** be the string "sat" (lowercase). "sat" denotes satoshis, the base unit used for Lightning/Bitcoin amounts.

description **OPTIONAL**. A human-readable memo describing the resource or service being paid for. This value is used as the description field of the BOLT11 invoice ([BOLT11]) and is distinct from any `description` auth-param that the base [I-D.httpauth-payment] scheme may include at the header level.

recipient **OPTIONAL**. Payment recipient in method-native format, per [I-D.payment-intent-charge]. Lightning implementations typically do not use this field; the invoice payee is implied by the BOLT11 invoice.

externalId **OPTIONAL**. Merchant's reference (e.g., order ID, invoice number), per [I-D.payment-intent-charge]. May be used for reconciliation or idempotency.

7.2. Method Details

The following fields are nested under `methodDetails` in the request JSON. The BOLT11 invoice (`methodDetails.invoice`) is the authoritative source for payment parameters. The `paymentHash` and `network` fields are provided as convenience to spare clients from decoding the invoice; if present, they **MUST** exactly match the values encoded in the invoice. Servers **MUST** verify this consistency before issuing the challenge. Clients **MUST** decode and verify the invoice independently before paying, and **MUST** reject challenges where the convenience fields do not match the invoice.

invoice **REQUIRED**. The full BOLT11-encoded payment request string (e.g., "lnbc100n1..."). This field is authoritative; all other payment parameters are derived from it.

paymentHash **OPTIONAL** convenience field. The payment hash embedded in the invoice, as a lowercase hex-encoded string. If present, **MUST** equal the payment hash decoded from `invoice`.

network **OPTIONAL** convenience field. Identifies the Lightning Network the invoice is issued on. **MUST** be one of "mainnet", "regtest", or "signet". If present, **MUST** match the network implied by the invoice's human-readable prefix. Defaults to "mainnet" if omitted. Clients **SHOULD** reject invoices whose network does not match their configured network.

8. Credential Schema

The `Authorization` header carries a single base64url-encoded JSON token (no auth-params). The decoded object contains two top-level fields:

challenge **REQUIRED**. An echo of the challenge auth-params from the `WWW-Authenticate` header: `id`, `realm`, `method`, `intent`, `request`, and (if present) `expires`. This binds the credential to the exact challenge that was issued.

source **OPTIONAL**. A payer identifier string, as defined by [I-D.httpauth-payment]. The **RECOMMENDED** format is a Decentralized Identifier (DID) per [W3C-DID]. Lightning-specific implementations **MAY** omit this field; servers **MUST NOT** require it.

payload **REQUIRED**. A JSON object containing the Lightning-specific credential fields. The single required field is `preimage`: the 32-byte payment preimage revealed upon successful HTLC settlement, encoded as a lowercase hex string.

Example (decoded):

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "lightning",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2026-03-15T12:05:00Z"
  },
  "source": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK",
  "payload": {
    "preimage": "a3f1...e209"
  }
}
```

9. Verification Procedure

Upon receiving a request with a credential, the server **MUST**:

1. Decode the base64url credential and parse the JSON.

2. Verify that "preimage" is present and is a 64-character lowercase hex string.
3. Look up the stored challenge using `credential.challenge.id`. Retrieve the `paymentHash` recorded when the challenge was issued. If no matching challenge is found, reject the request.
4. Verify that all fields in `credential.challenge` exactly match the stored challenge auth-params (e.g., `id`, `realm`, `method`, `intent`, `request`, `expires`).
5. Decode the echoed `credential.challenge.request` and verify that the `amount` and `currency` in it match the invoice parameters stored with the challenge.
6. Compute sha256(hex_to_bytes(preimage)) and verify the result equals the stored paymentHash.

9.1. Challenge Binding

To prevent preimage replay across different resources or sessions, the server **MUST** store the issued `paymentHash` keyed by the challenge `id` assigned at challenge time. When the client presents a credential, the server **MUST** verify `credential.challenge` is an exact echo of the issued challenge params and look up the stored `paymentHash` by `credential.challenge.id`. A preimage that is valid for one challenge **MUST NOT** be accepted for a different challenge.

10. Settlement Procedure

Lightning Network settlement is synchronous from the payer's perspective: the preimage is only revealed after the HTLC resolves (see [BOLT4]). Settlement is therefore considered complete at the moment the server successfully verifies the preimage (step 6 of Section 9). No out-of-band confirmation is required.

The server **MUST** atomically mark the challenge as consumed and deliver the resource. Specifically, the challenge invalidation and the decision to return HTTP 200 **MUST** be treated as a single operation: a challenge that has been marked consumed **MUST NOT** be accepted again, even if the resource delivery subsequently fails. If resource delivery fails after the challenge is consumed, the server **MUST** return an appropriate HTTP error (e.g., 500) and **MUST NOT** reissue the same challenge. The client **MUST** treat such a response as a payment loss and **MAY** retry with a new payment. Unlike reversible payment methods, Lightning settlement is final once the preimage is revealed; the payment cannot be refunded through the payment channel.

Servers **MUST** include `Cache-Control: no-store` on all HTTP 402 responses. The challenge contains a single-use invoice; caching it could cause clients to attempt to pay a stale or already-settled invoice.

10.1. Challenge Expiry and Invoice Expiry

A challenge has two independent expiry signals: the `expires` auth-param on the `WWW-Authenticate` header (defined by [I-D.httpauth-payment]) and the expiry field embedded in the BOLT11 invoice. The effective expiry of a challenge is the earlier of the two.

Servers **MUST NOT** set the `expires` auth-param to a time later than the invoice's BOLT11 expiry time. Clients **SHOULD** use the earlier of the two values when deciding whether to attempt payment. Servers **MUST** reject credentials for challenges whose effective expiry has passed, regardless of which signal triggered it.

10.2. Receipt Generation

The server **MUST** include a Payment-Receipt header in the 200 response with the following fields:

method **REQUIRED**. The string "lightning".

challengeId **REQUIRED**. The challenge identifier (the `id` from the WWW-Authenticate challenge) for audit and traceability correlation.

reference **REQUIRED**. The payment hash (SHA-256 of the preimage) as a lowercase hex string. Serves as a globally unique, publicly shareable payment receipt identifier. The preimage **MUST NOT** be used here, as it is a bearer secret and its exposure in logs, analytics, or shared receipts would allow replay.

status **REQUIRED**. The string "success".

timestamp **REQUIRED**. The settlement time in [RFC3339] format.

Example (decoded):

```
{
  "method": "lightning",
  "challengeId": "kM9xPqWvT2nJrHsY4aDfEb",
  "reference": "bc230847...",
  "status": "success",
  "timestamp": "2026-03-10T21:00:00Z"
}
```

11. Error Responses

When rejecting a credential, the server **MUST** return HTTP 402 (Payment Required) with a fresh `WWW-Authenticate: Payment` challenge per [I-D.httpauth-payment]. The server **SHOULD** include a response body conforming to RFC 9457 [RFC9457] Problem Details, with `Content-Type: application/problem+json`. The following problem types are defined for this intent:

https://paymentauth.org/problems/lightning/malformed-credential HTTP 402. The credential token could not be decoded, the JSON could not be parsed, or required fields (`challenge`, `payload`, `payload.preimage`) are absent or have the wrong type. A fresh challenge **MUST** be included in `WWW-Authenticate`.

https://paymentauth.org/problems/lightning/unknown-challenge HTTP 402. The value of `credential.challenge.id` does not match any challenge issued by this server, or the challenge has already been consumed. A fresh challenge **MUST** be included in `WWW-Authenticate`.

https://paymentauth.org/problems/lightning/invalid-preimage HTTP 402. `SHA-256(payload.preimage)` does not equal the `paymentHash` stored for the identified challenge. A fresh challenge **MUST** be included in `WWW-Authenticate`.

https://paymentauth.org/problems/lightning/expired-invoice HTTP 402. The BOLT11 invoice associated with the challenge has passed its expiry time, or the challenge `expires` auth-param indicates the challenge has expired. A fresh challenge **MUST** be included in `WWW-Authenticate`.

Example error response body:

```
{
  "type": "https://paymentauth.org/problems/lightning/invalid-preimage",
  "title": "Invalid Preimage",
  "status": 402,
  "detail": "SHA-256 of the provided preimage does not match the stored
payment hash"
}
```

12. Security Considerations

12.1. Preimage Uniqueness

Each BOLT11 invoice **MUST** use a freshly generated random payment hash. Reusing a payment hash allows a client who has previously paid to replay the credential indefinitely.

12.2. Amount Verification

The server **MUST** verify that the amount encoded in the BOLT11 invoice matches the intended charge amount before issuing the challenge. Clients **SHOULD** independently decode and verify the invoice amount before paying.

12.3. Invoice Expiry

BOLT11 invoices carry an expiry field (default 3600 seconds). Servers **MUST NOT** accept credentials for expired invoices. Servers **MAY** enforce a shorter expiry window to reduce the window in which a compromised preimage could be replayed.

12.4. Preimage Confidentiality

The payment preimage **MUST** only be transmitted over HTTPS. Servers, clients, and intermediaries **MUST NOT** log, persist, or include preimages in error responses, analytics, or diagnostic output. Exposure of the preimage allows any party to present it as a valid credential until the challenge has been consumed or the invoice has expired. Servers **MUST** invalidate a challenge on first successful use to enforce consume-once semantics. The acceptance check and invalidation **MUST** be atomic: concurrent requests presenting the same valid preimage **MUST** result in exactly one success and one rejection, with no window in which both are accepted.

13. IANA Considerations

13.1. Payment Method Registration

This document requests registration of the following entry in the "HTTP Payment Methods" registry established by [I-D.httpauth-payment]:

Method Identifier	Description	Reference
`lightning`	Lightning Network BOLT11 invoice payment	This document

Table 1

Contact: Lightspark (contact@lightspark.com)

13.2. Payment Intent Registration

This document requests registration of the following entry in the "HTTP Payment Intents" registry established by [I-D.httpauth-payment]:

Intent	Applicable Methods	Description	Reference
`charge`	`lightning`	One-time BOLT11 invoice payment gating access to a resource	This document

Table 2

14. References

14.1. 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>.

[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>.

[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>.

[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>.

[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>.

[I-D.payment-intent-charge] Moxey, J., Ryan, B., and T. Meagher, "'charge' Intent for HTTP Payment Authentication", 2026, <https://datatracker.ietf.org/doc/draft-payment-intent-charge/>.

[BOLT11] Lightning Network Developers, "BOLT #11: Invoice Protocol for Lightning Payments", 2024, <https://github.com/lightning/bolts/blob/master/11-payment-encoding.md>.

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

14.2. Informative References

[BOLT4] Lightning Network Developers, "BOLT #4: Onion Routing Protocol", 2024, <https://github.com/lightning/bolts/blob/master/04-onion-routing.md>.

[W3C-DID] W3C, "Decentralized Identifiers (DIDs) v1.0", 2022, <https://www.w3.org/TR/did-core/>.

Appendix A. Examples

A.1. Initial Request and 402 Challenge

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

HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment id="kM9xPqWvT2nJrHsY4aDfEb",
  realm="api.example.com",
  method="lightning",
  intent="charge",

request="eyJhbW91bnQiOiIxMDAiLCJjdXJyZW5jeSI6IkJUQyIsImRlc2NyaXB0aW9uIjoiV2Vh
dGhlciByZXBvcnQgZm9yIDk0MTA3IiwibWV0aG9kRGV0YWlscyI6eyJpbnZvaWNlIjoibG5iYzF1M
XAuLi4iLCJwYXltZW50SGFzaCI6ImJjMjMwODQ3Li4uIiwibmV0d29yayI6Im1haW5uZXQifX0",
  expires="2026-03-15T12:05:00Z"
Cache-Control: no-store
```

Decoded `request`:

```
{
  "amount": "100",
  "currency": "sat",
  "description": "Weather report for 94107",
  "methodDetails": {
    "invoice": "lnbc1u1p...",
    "paymentHash": "bc230847...",
    "network": "mainnet"
  }
}
```

A.2. Retry with Credential

```
GET /weather HTTP/1.1
Host: api.example.com
Authorization: Payment
eyJjaGFsbGVuZ2UiOnsiaWQiOiJrTTl4UHFXdlQybkpySHNZNGFEZkViIiwicmVhbG0iOiJhcGkuZ
XhhbXBsZS5jb20iLCJtZXRob2QiOiJsaWdodG5pbmciLCJpbnRlbnQiOiJjaGFyZ2UiLCJyZXF1ZX
N0IjoiZXlKLi4uIiwiZXhwaXJlcyI6IjIwMjYtMDMtMTVUMTI6MDU6MDBaIn0sInBheWxvYWQiOns
icHJlaW1hZ2UiOiJhM2YxLi4uZTIwOSJ9fQ

HTTP/1.1 200 OK
Payment-Receipt:
eyJtZXRob2QiOiJsaWdodG5pbmciLCJyZWZlcmVuY2UiOiJhM2YxLi4uZTIwOSIsInN0YXR1cyI6I
nN1Y2Nlc3MiLCJ0aW1lc3RhbXAiOiIyMDI2LTAzLTEwVDIxOjAwOjAwWiJ9
Content-Type: application/json

{"temperature": 72, "condition": "sunny"}
```

Decoded receipt:

```
{
  "method": "lightning",
  "challengeId": "kM9xPqWvT2nJrHsY4aDfEb",
  "reference": "bc230847...",
  "status": "success",
  "timestamp": "2026-03-10T21:00:00Z"
}
```

Decoded credential:

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "lightning",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2026-03-15T12:05:00Z"
  },
  "source": "did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK",
  "payload": {
    "preimage": "a3f1...e209"
  }
}
```

Appendix B. Acknowledgements

The authors thank the Spark SDK team and the broader Lightning Network developer community. The authors also thank Brendan Ryan for his review of this document.

Authors' Addresses

Kevin Zhang
Lightspark
Email: kevz@lightspark.com

Jeremy Klein
Lightspark
Email: jeremy@lightspark.com

Zhen Lu
Lightspark
Email: zhenlu@lightspark.com