
Workgroup: Network Working Group
Internet-Draft: draft-stellar-charge-00
Published: 18 May 2026
Intended Status: Informational
Expires: 19 November 2026
Author: M. Salloum
Stellar Development Foundation

Stellar Charge Intent for HTTP Payment Authentication

Abstract

This document defines the "charge" intent for the "stellar" payment method in the Payment HTTP Authentication Scheme. It specifies how clients and servers exchange one-time [SEP-41] token transfers on the Stellar blockchain, with optional server-sponsored transaction fees.

Two credential types are supported: `type="transaction"` (default), where the client sends the signed transaction to the server for submission, and `type="hash"` (fallback), where the client broadcasts the transaction directly to the network and presents the on-chain transaction hash for server verification.

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.

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	4
1.1. Pull Mode (Default)	4
1.2. Push Mode (Fallback)	4
1.3. Relationship to the Charge Intent	5
2. Requirements Language	5
3. Terminology	5
4. Request Schema	6
4.1. Shared Fields	6
4.2. Method Details	7
5. Credential Schema	7
5.1. Credential Structure	8
5.2. Transaction Payload — Pull Mode (type="transaction")	8
5.3. Hash Payload — Push Mode (type="hash")	9
6. Ledger Expiration	10
7. Fee Payment	10
7.1. Pull Mode	10
7.1.1. Server-Sponsored Fees	10
7.1.2. Client-Paid Fees	11
7.2. Push Mode	11
8. Verification	11
8.1. Pull Mode Verification	12
8.1.1. Sponsored Flow Checks	12
8.1.2. Unsponsored Flow Checks	12

8.2. Push Mode Verification	13
9. Error Codes	13
10. Settlement Procedure	13
10.1. Pull Mode Settlement — Sponsored	13
10.2. Pull Mode Settlement — Un-sponsored	14
10.3. Push Mode Settlement (type="hash")	14
10.4. Receipt	14
11. Security Considerations	15
11.1. Facilitator Safety	15
11.2. Replay Protection	15
11.2.1. Pull Mode	15
11.2.2. Push Mode	15
11.3. Amount and Asset Verification	15
11.4. Simulation Integrity	15
11.5. Fee Exhaustion	15
12. IANA Considerations	16
12.1. Payment Method Registration	16
12.2. Payment Intent Registration	16
13. References	16
13.1. Normative References	16
13.2. Informative References	17
Appendix A. ABNF Collected	18
Appendix B. Examples	18
B.1. Pull Mode — Sponsored (type="transaction")	18
B.2. Pull Mode — Un-sponsored (type="transaction")	19
B.3. Push Mode (type="hash")	20
Appendix C. Acknowledgements	21
Author's Address	21

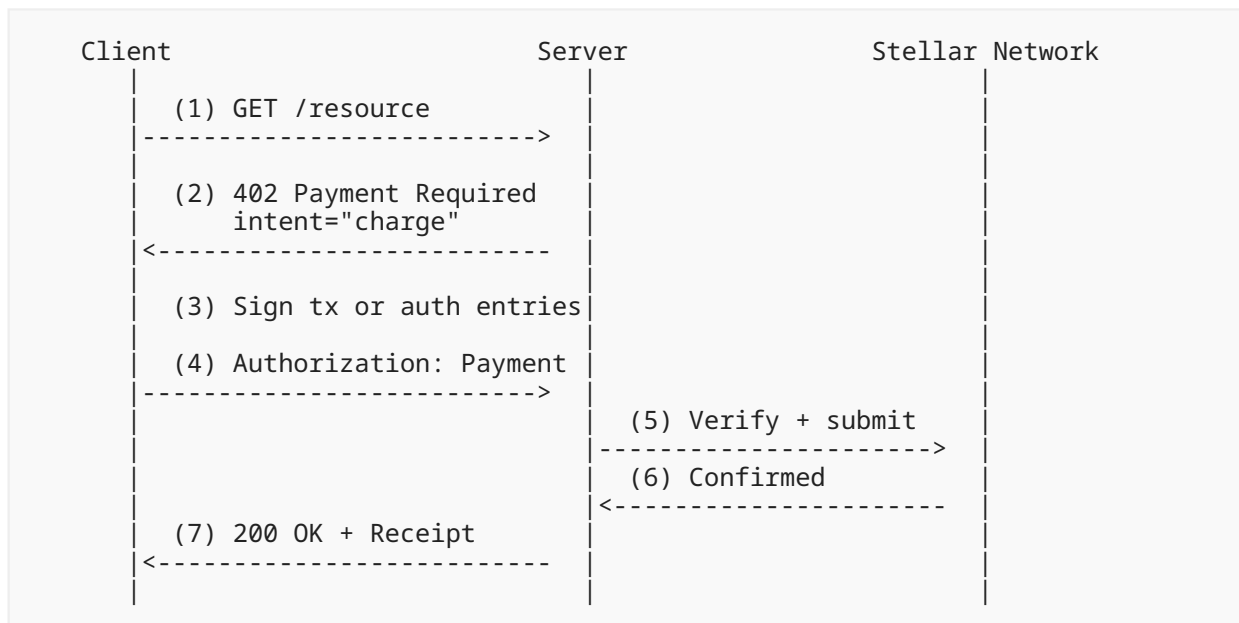
1. Introduction

The charge intent represents a one-time payment of a specified amount, as defined in [I-D.payment-intent-charge]. The server may settle the payment any time before the challenge expires auth-param timestamp.

This document specifies how to implement the charge intent using SEP-41 [SEP-41] tokens on the Stellar smart contract platform. [SEP-41] defines a standard token interface for Stellar smart contracts, including Stellar Asset Contracts (SAC) [SAC] and custom token implementations.

1.1. Pull Mode (Default)

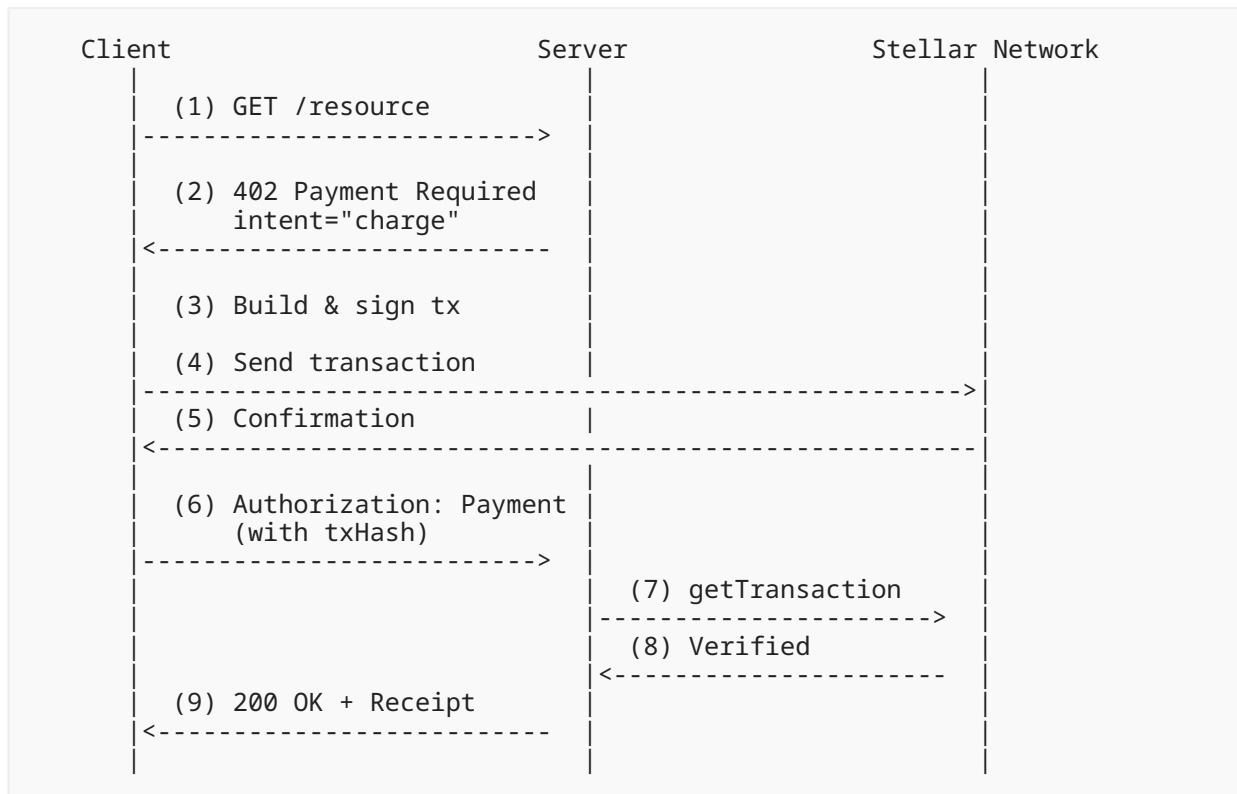
The default flow, called "pull mode", uses type="transaction" credentials. The client signs the transaction (or authorization entries) and the server "pulls" it for submission to the Stellar network:



In this model the server controls transaction submission, enabling fee sponsorship (Section 7.1.1) and server-side retry logic. When feePayer is true, step (3) signs only authorization entries and step (5) includes the server rebuilding the transaction as source. When feePayer is false, step (3) builds and signs a complete transaction and the server submits it without modification.

1.2. Push Mode (Fallback)

The fallback flow, called "push mode", uses type="hash" credentials. The client "pushes" the transaction to the network itself and presents the confirmed transaction hash:



This flow is useful when the client cannot or does not wish to delegate submission to the server. The server verifies the payment by fetching and inspecting the on-chain transaction via RPC.

1.3. 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 Stellar-specific methodDetails, payload, and verification procedures for the "stellar" 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

SEP-41 Token A Stellar smart contract implementing the [SEP-41] token interface, exposing transfer, balance, and related functions. Identified by a C-prefixed Stellar smart contract address. Stellar Asset Contracts (SAC) [SAC] are a common [SEP-41] implementation that wrap classic Stellar assets.

Authorization Entry A signed data structure scoping a Stellar smart contract invocation to a specific invoker and ledger sequence range. See [\[STELLAR-AUTH\]](#).

Fee Sponsorship An arrangement where the server pays Stellar network fees on behalf of the client. The client signs only authorization entries; the server acts as the transaction source account. Servers **MAY** additionally wrap the rebuilt transaction in a fee bump transaction to adjust fees without invalidating the client's authorization entries.

DEFAULT_LEDGER_CLOSE_TIME The normative fallback value for the average Stellar ledger close time: 5 seconds. Used to convert wall-clock expiry to a ledger sequence number when network-provided estimates are unavailable or impractical.

DEFAULT_CHALLENGE_EXPIRY The normative fallback challenge expiry duration: 5 minutes. Used when the expires auth-param is absent from the challenge.

CAIP-2 Network Identifier A chain identifier per the CAIP-2 Stellar namespace [\[CAIP-2-STELLAR\]](#) (e.g., stellar:pubnet, stellar:testnet).

Pull Mode The default settlement flow where the client signs the transaction (or authorization entries) and the server submits it (type="transaction"). The server "pulls" the signed transaction from the credential. Enables fee sponsorship and server-side retry logic.

Push Mode The fallback settlement flow where the client broadcasts the transaction itself and presents the confirmed transaction hash (type="hash"). The client "pushes" the transaction to the network directly. Cannot be used with fee sponsorship.

4. Request Schema

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

This specification implements the shared request fields defined in [\[I-D.payment-intent-charge\]](#).

4.1. Shared Fields

Field	Type	Presence	Description
amount	string	REQUIRED	Stringified non-negative integer in the [SEP-41] token's base units (e.g., "100000" for 0.01 USDC with 7 decimals)
currency	string	REQUIRED	[SEP-41] token contract address (C-prefixed Stellar smart contract ID)
recipient	string	REQUIRED	Stellar account address of the payment recipient

Field	Type	Presence	Description
description	string	OPTIONAL	Human-readable payment description
externalId	string	OPTIONAL	Merchant reference (order ID, invoice number, etc.)

Table 1

Challenge expiry is conveyed by the expires auth-param in WWW-Authenticate per [I-D.httpauth-payment].

4.2. Method Details

Field	Type	Presence	Description
methodDetails.network	string	REQUIRED	CAIP-2 Stellar chain identifier (stellar:pubnet or stellar:testnet)
methodDetails.feePayer	boolean	OPTIONAL	If true, server pays transaction fees (default: false)

Table 2

If methodDetails.feePayer is true, the server sponsors transaction fees per Section 7.1.1. If false or omitted, the client **MUST** build a fully signed, network-ready transaction per Section 7.1.2. Fee sponsorship is only available in pull mode (type="transaction"); push mode (type="hash") **MUST NOT** be used with feePayer: true.

Example:

```
{
  "amount": "10000000",
  "currency": "CBIELTK6YBZJU5UP2WWQEUCYKLPUGAUNZ2BQ4W",
  "recipient": "GBHEGW3KWOY2OFH767EDALFGCUTBOEVBDQMCKU",
  "description": "API access fee",
  "methodDetails": {
    "network": "stellar:testnet",
    "feePayer": true
  }
}
```

5. Credential Schema

The credential in the Authorization header contains a base64url-encoded JSON object per [I-D.httpauth-payment].

5.1. Credential Structure

Field	Type	Presence	Description
challenge	object	REQUIRED	Echo of the challenge auth-params from WWW-Authenticate per [I-D.httpauth-payment]
payload	object	REQUIRED	Stellar-specific payload
source	string	OPTIONAL	Payer DID

Table 3

The source field, if present, **SHOULD** use the did:pkh method [DID-PKH] with the CAIP-2 network identifier and the payer's Stellar address (e.g., did:pkh:stellar:testnet:GABC...).

5.2. Transaction Payload — Pull Mode (type="transaction")

Field	Type	Presence	Description
type	string	REQUIRED	"transaction"
transaction	string	REQUIRED	Base64-encoded XDR

Table 4

`transaction` Base64-encoded XDR of a Stellar transaction as defined in [STELLAR-XDR]. The `transaction` **MUST** contain exactly one operation of type `invokeHostFunction`.

When `feePayer` is `true`: the transaction source account **MUST** be set to the all-zeros Stellar account (GAAWHF). The server replaces it with its own address at settlement. Authorization entries **MUST** be signed by the client.

When `feePayer` is `false`: the transaction **MUST** be fully signed and network-ready, including valid sequence number, fee, `timeBounds`, and source account. The server **MUST** submit this transaction without modification.

Example:

```

{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "stellar",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "type": "transaction",
    "transaction": "AAAAAgAAAAABriIN4..."
  },
  "source": "did:pkh:stellar:testnet:GABC..."
}

```

5.3. Hash Payload — Push Mode (type="hash")

In push mode (type="hash"), the client has already broadcast the transaction to the Stellar network. The hash field contains the transaction hash for the server to verify on-chain.

Field	Type	Presence	Description
type	string	REQUIRED	"hash"
hash	string	REQUIRED	Stellar transaction hash (64-character hex string)

Table 5

Push mode **MUST NOT** be used when `feePayer` is `true` in the challenge request. Since the client has already broadcast the transaction, the server cannot act as fee sponsor. Servers **MUST** reject `type="hash"` credentials when the challenge specifies `feePayer: true`.

Example:

```
{
  "challenge": {
    "id": "pT7yHnKmQ2wErXsZ5vCbN1",
    "realm": "api.example.com",
    "method": "stellar",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "type": "hash",
    "hash":
"a1b2c3d4e5f6789012345678901234567890123456789012345678901234abcd"
  },
  "source": "did:pkh:stellar:testnet:GABC..."
}
```

6. Ledger Expiration

Stellar uses ledger sequence numbers for transaction and authorization entry expiration rather than wall-clock timestamps. Clients **MUST** derive the ledger expiration from the challenge `expires` auth-param as follows.

If `expires` is absent, clients **SHOULD** default to `DEFAULT_CHALLENGE_EXPIRY` (5 minutes) from the current time.

```
ledgerExpiration =
  currentLedger +
  ceil((expires - now) / DEFAULT_LEDGER_CLOSE_TIME)
```

where `DEFAULT_LEDGER_CLOSE_TIME` is 5 seconds. `currentLedger` **MUST** be obtained from the Stellar network via Stellar RPC `getLatestLedger` [[STELLAR-RPC](#)].

When `feePayer` is true, clients **MUST** set the authorization entry expiration ledger to this value.

When `feePayer` is false, clients **MUST** set the transaction `timeBounds.maxTime` to the `expires` timestamp. The transaction **MUST NOT** be valid beyond the challenge expiry.

7. Fee Payment

7.1. Pull Mode

7.1.1. Server-Sponsored Fees

When `methodDetails.feePayer` is true:

1. The client obtains `currentLedger` via Stellar RPC `getLatestLedger` and computes the authorization entry expiration per [Section 6](#).

2. The client builds an `invokeHostFunction` transaction with the all-zeros source account, containing a single operation calling `transfer(from, to, amount)` on the [SEP-41] token contract. The client simulates the transaction to identify the required authorization entries.
3. The client signs the authorization entries using credential type `sorobanCredentialsAddress`. The client **MUST NOT** sign the full transaction.
4. The client encodes the transaction with signed authorization entries as base64 XDR and places it in `payload.transaction`.
5. Upon receiving the credential, the server verifies it per [Section 8](#), rebuilds the transaction with itself as source account, and submits it per [Section 10](#).

Servers acting as fee sponsors:

- **MUST** maintain sufficient XLM balance to cover fees.
- **MAY** reject new challenges when XLM balance is below a safe operational threshold.

7.1.2. Client-Paid Fees

When `methodDetails.feePayer` is `false` or absent:

1. The client sets `timeBounds.maxTime` to the `expires auth-param` value, or `DEFAULT_CHALLENGE_EXPIRY` from the current time if absent. The transaction **MUST NOT** be valid beyond the challenge expiry. See [Section 6](#).
2. The client builds a fully signed `invokeHostFunction` transaction containing a single operation calling `transfer(from, to, amount)` on the [SEP-41] token contract, including sequence number, fee, and `timeBounds`.
3. The client encodes the complete, signed transaction as base64 XDR in `payload.transaction`.
4. Upon receiving the credential, the server verifies it per [Section 8](#) and submits the transaction without modification per [Section 10](#).

7.2. Push Mode

In push mode, the client builds, signs, and broadcasts the transaction independently. The client pays all fees. Fee sponsorship is not available in push mode.

8. Verification

Before settling a charge credential, servers **MUST** first validate that `payload.type` is `"transaction"` or `"hash"`, then proceed with the appropriate verification path. If any check fails, the server **MUST** return a `verification-failed` error per [\[I-D.httpauth-payment\]](#).

If the Stellar RPC is unavailable for a required simulation step, servers **MUST** treat this as a server error (HTTP 5xx) rather than a `verification-failed` response, and **MUST NOT** settle the credential.

8.1. Pull Mode Verification

8.1.1. Sponsored Flow Checks

1. The challenge id matches an outstanding, unsettled challenge issued by this server, and the current time is before the challenge expires auth-param.
2. The decoded transaction contains exactly one `invokeHostFunction` operation with function type `hostFunctionTypeInvokeContract`.
3. The invoked function is `transfer(from, to, amount)` on the contract matching currency. The to argument **MUST** equal recipient and the amount argument **MUST** equal amount (as `i128`) from the challenge request.
4. The transaction's network passphrase **MUST** correspond to `methodDetails.network`.
5. The server **MUST** simulate the transaction via Stellar RPC. The simulation **MUST** succeed and **MUST** emit events showing only the expected balance changes: a decrease of amount for the payer and an increase of amount for the recipient. Any other balance change **MUST** cause verification to fail.
6. The transaction source account is the all-zeros account (GAAA`WHF`).
7. Authorization entries **MUST** use credential type `sorobanCredentialsAddress` only, and **MUST NOT** contain subInvocations beyond the single [\[SEP-41\]](#) token transfer.
8. The authorization entry expiration **MUST NOT** exceed `currentLedger + ceil((expires - now) / DEFAULT_LEDGER_CLOSE_TIME)`.
9. The server's address **MUST NOT** appear as the from argument or in any authorization entry.

8.1.2. Unsponsored Flow Checks

1. The challenge id matches an outstanding, unsettled challenge issued by this server, and the current time is before the challenge expires auth-param.
2. The decoded transaction contains exactly one `invokeHostFunction` operation with function type `hostFunctionTypeInvokeContract`.
3. The invoked function is `transfer(from, to, amount)` on the contract matching currency. The to argument **MUST** equal recipient and the amount argument **MUST** equal amount (as `i128`) from the challenge request.
4. The transaction's network passphrase **MUST** correspond to `methodDetails.network`.
5. The server **MUST** simulate the transaction via Stellar RPC. The simulation **MUST** succeed and **MUST** emit events showing only the expected balance changes: a decrease of amount for the payer and an increase of amount for the recipient. Any other balance change **MUST** cause verification to fail.
6. `timeBounds.maxTime` **MUST NOT** exceed the expires timestamp from the challenge.

8.2. Push Mode Verification

For push mode credentials (`type="hash"`), the server **MUST** fetch the transaction via Stellar RPC `getTransaction` [STELLAR-RPC] and verify:

1. The challenge id matches an outstanding, unsettled challenge issued by this server.
2. The transaction hash has not been previously consumed (see [Section 11.2](#)).
3. The transaction exists and has status `SUCCESS`.
4. The transaction contains exactly one `invokeHostFunction` operation calling `transfer(from, to, amount)` on the contract matching currency. The `to` argument **MUST** equal recipient and the `amount` argument **MUST** equal amount (as `i128`) from the challenge request.
5. Mark the transaction hash as consumed.

9. Error Codes

This specification defines the following additional error code beyond those in [I-D.httpauth-payment]:

Code	HTTP	Description
<code>settlement-failed</code>	402	Credential valid but on-chain settlement failed

Table 6

Servers **MUST** return `settlement-failed` when a credential passes verification but the Stellar transaction fails on-chain (e.g., insufficient funds or sequence number conflict). This is distinct from `verification-failed`, which indicates the credential failed validation checks.

10. Settlement Procedure

10.1. Pull Mode Settlement — Sponsored

1. Parse the base64 XDR transaction from `payload.transaction`.
2. Extract all operations and authorization entries.
3. Rebuild a new transaction with:
 - Source account: the server's Stellar address.
 - Operations: copied from the client's transaction.
 - Authorization entries: copied from the client's transaction.
4. Sign the rebuilt transaction with the server's key.
5. Submit via Stellar RPC `sendTransaction` [STELLAR-RPC].
6. Verify the submission returns `PENDING` status, then poll until `SUCCESS` or `FAILED`.

7. On SUCCESS, return a receipt per [Section 10.4](#). On FAILED, return a settlement-failed error per [Section 9](#).

10.2. Pull Mode Settlement — Un-sponsored

1. Verify the transaction per [Section 8](#).
2. Submit the received transaction as-is via Stellar RPC `sendTransaction` [[STELLAR-RPC](#)]. The server **MUST NOT** modify the transaction.
3. Poll until SUCCESS or FAILED.
4. On SUCCESS, return a receipt per [Section 10.4](#). On FAILED, return a settlement-failed error per [Section 9](#).

10.3. Push Mode Settlement (type="hash")

For push mode credentials, the client has already broadcast the transaction. The server checks the transaction hash against consumed hashes per [Section 11.2.2](#), verifies the transaction on-chain per [Section 8.2](#), and returns a receipt per [Section 10.4](#).

Limitations:

- **MUST NOT** be used with `feePayer: true` (client must pay their own fees)
- Server cannot modify or enhance the transaction

10.4. Receipt

Upon successful settlement, servers **MUST** return a `Payment-Receipt` header per [[I-D.httpauth-payment](#)].

The receipt payload fields:

Field	Type	Presence	Description
method	string	REQUIRED	"stellar"
reference	string	REQUIRED	Transaction hash
status	string	REQUIRED	"success"
timestamp	string	REQUIRED	[RFC3339] settlement time
externalId	string	OPTIONAL	Echoed from request

Table 7

11. Security Considerations

11.1. Facilitator Safety

When `feePayer` is `true`, the server rebuilds and signs the transaction. A malicious client could craft a transaction to drain the server's account.

Servers **MUST** verify their own address does not appear as the `from` transfer argument or in any authorization entry before signing. Servers **MUST** re-simulate the rebuilt transaction and **MUST** reject any credential whose simulation emits unexpected balance changes.

11.2. Replay Protection

11.2.1. Pull Mode

Authorization entry expiration (keyed to ledger sequence) and Stellar sequence number consumption prevent transaction replay. Servers **MUST** reject credentials referencing an expired or already-settled challenge id.

11.2.2. Push Mode

Servers **MUST** maintain a set of consumed transaction hashes. Before accepting a push mode credential, the server **MUST** check whether the hash has already been consumed and reject the credential if it has. After successful verification, the server **MUST** atomically mark the hash as consumed.

11.3. Amount and Asset Verification

Clients **MUST** decode and verify the challenge request before signing. Clients **MUST** verify that amount, currency, and recipient match their expectations prior to authorizing any transfer.

11.4. Simulation Integrity

The simulation requirement in [Section 8](#) ensures the transaction behaves as specified. Servers **MUST** treat any unexpected balance change as a verification failure, regardless of whether it favors the server or a third party.

11.5. Fee Exhaustion

Servers offering fee sponsorship are exposed to denial-of-service attacks where adversaries submit valid-looking credentials that fail on-chain, causing the server to pay fees without receiving payment. Servers **SHOULD** implement rate limiting and **MAY** require client authentication before issuing sponsored challenges.

12. IANA Considerations

12.1. 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
stellar	Stellar [SEP-41] token transfer	This document

Table 8

Contact: Stellar Development Foundation (developers@stellar.org)

12.2. 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	stellar	One-time [SEP-41] token transfer	This document

Table 9

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

13.2. Informative References

- [SEP-41]** Stellar Development Foundation, "SEP-41: Token Interface", 2024, <<https://stellar.org/protocol/sep-41>>.
- [CAIP-2-STELLAR]** Chain Agnostic Standards Alliance, "CAIP-2: Stellar Namespace", 2023, <<https://namespaces.chainagnostic.org/stellar/caip2>>.
- [STELLAR-AUTH]** Stellar Development Foundation, "Stellar Contracts Authorization Framework", n.d., <<https://developers.stellar.org/docs/learn/encyclopedia/security/authorization>>.
- [SAC]** Stellar Development Foundation, "Stellar Asset Contract", n.d., <<https://developers.stellar.org/docs/tokens/stellar-asset-contract>>.
- [DID-PKH]** W3C CCG, "did:pkh Method Specification", n.d., <<https://github.com/w3c-ccg/did-pkh>>.
- [STELLAR-XDR]** Stellar Development Foundation, "Stellar XDR Definitions", n.d., <<https://github.com/stellar/stellar-xdr>>.
- [STELLAR-RPC]** Stellar Development Foundation, "Stellar RPC Reference", n.d., <<https://developers.stellar.org/docs/data/rpc/api-reference>>.

Appendix A. ABNF Collected

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

stellar-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. Pull Mode — Sponsored (type="transaction")

Challenge:

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment
  id="kM9xPqWvT2nJrHsY4aDfEb",
  realm="api.example.com",
  method="stellar",
  intent="charge",
  request="eyJhbW91bnQiOiIxMDAwMDAwMCI6ImN1cnJlb...",
  expires="2025-02-05T12:05:00Z"
Cache-Control: no-store
```

The request decodes to:

```
{
  "amount": "10000000",
  "currency": "CBIELTK6YBZJU5UP2WWQEUCYKLPUGAUNZ2BQ4W",
  "recipient": "GBHEGW3KWOY20FH767EDALFGCUTBOEVBDQMCKU",
  "methodDetails": {
    "network": "stellar:testnet",
    "feePayer": true
  }
}
```

This requests a transfer of 1.0 USDC (10000000 base units, assuming 7 decimal places).

Credential:

```
GET /api/resource HTTP/1.1
Host: api.example.com
Authorization: Payment eyJjaGFsbGVuZ2Ui...
```

Decoded credential:

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "stellar",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "type": "transaction",
    "transaction": "AAAAAgAAAABriIN4..."
  },
  "source": "did:pkh:stellar:testnet:GABC..."
}
```

Receipt:

```
{
  "method": "stellar",
  "reference":
"a1b2c3d4e5f6789012345678901234567890123456789012345678901234abcd",
  "status": "success",
  "timestamp": "2025-02-05T12:04:32Z"
}
```

B.2. Pull Mode — Un-sponsored (type="transaction")

Challenge:

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment
  id="pT7yHnKmQ2wErXsZ5vCbN1",
  realm="api.example.com",
  method="stellar",
  intent="charge",
  request="eyJhbW91bnQiOiIxMDAwMDAwMCIImN1cnJlb...",
  expires="2025-02-05T12:05:00Z"
Cache-Control: no-store
```

The request decodes to:

```
{
  "amount": "10000000",
  "currency": "CBIELTK6YBZJU5UP2WWQEUCYKLP6AUNZ2BQ4W",
  "recipient": "GBHEGW3KWOY20FH767EDALFGCUTBOEVBDQMCKU",
  "methodDetails": {
    "network": "stellar:testnet",
    "feePayer": false
  }
}
```

Credential:

```
GET /api/resource HTTP/1.1
Host: api.example.com
Authorization: Payment eyJjaGFsbGVuZ2Ui...
```

Decoded credential:

```
{
  "challenge": {
    "id": "pT7yHnKmQ2wErXsZ5vCbN1",
    "realm": "api.example.com",
    "method": "stellar",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "type": "transaction",
    "transaction": "AAAAAgAAAABriIN4..."
  },
  "source": "did:pkh:stellar:testnet:GABC..."
}
```

Receipt:

```
{
  "method": "stellar",
  "reference":
  "b2c3d4e5f6789012345678901234567890ab1234567890123456789012345678",
  "status": "success",
  "timestamp": "2025-02-05T12:04:41Z"
}
```

B.3. Push Mode (type="hash")

The client broadcasts the transaction itself and presents the confirmed hash. Cannot be used with fee sponsorship.

Credential:

```
{
  "challenge": { "...": "echoed challenge" },
  "payload": {
    "type": "hash",
    "hash":
    "a1b2c3d4e5f6789012345678901234567890123456789012345678901234abcd"
  }
}
```

Appendix C. Acknowledgements

The author thanks the Stellar community for their input and feedback on this specification.

Author's Address

Marcelo Salloum
Stellar Development Foundation
Email: marcelo@stellar.org