Workgroup:	Network Working Group		
Internet-Draft:	draft-tempo-charge-00		
Published:	9 September 2026		
Intended Status:	Informational		
Expires:	13 March 2027		
Authors:	J. Moxey	B. Ryan	T. Meagher
	Tempo Labs	*Tempo Labs*	*Tempo Labs*

Tempo Charge Intent for HTTP Payment Authentication

Abstract

This document defines the "charge" intent for the "tempo" payment method in the Payment HTTP Authentication Scheme [I-D.httpauth-payment]. It specifies how clients and servers exchange one-time TIP-20 token transfers on the Tempo blockchain.

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

Table of Contents

1. Introduction

The `charge` intent represents a one-time payment of a specified amount. The server may submit the signed transaction any time before the challenge `expires` auth-param timestamp.

This specification defines the request schema, credential formats, and settlement procedures for charge transactions on Tempo.

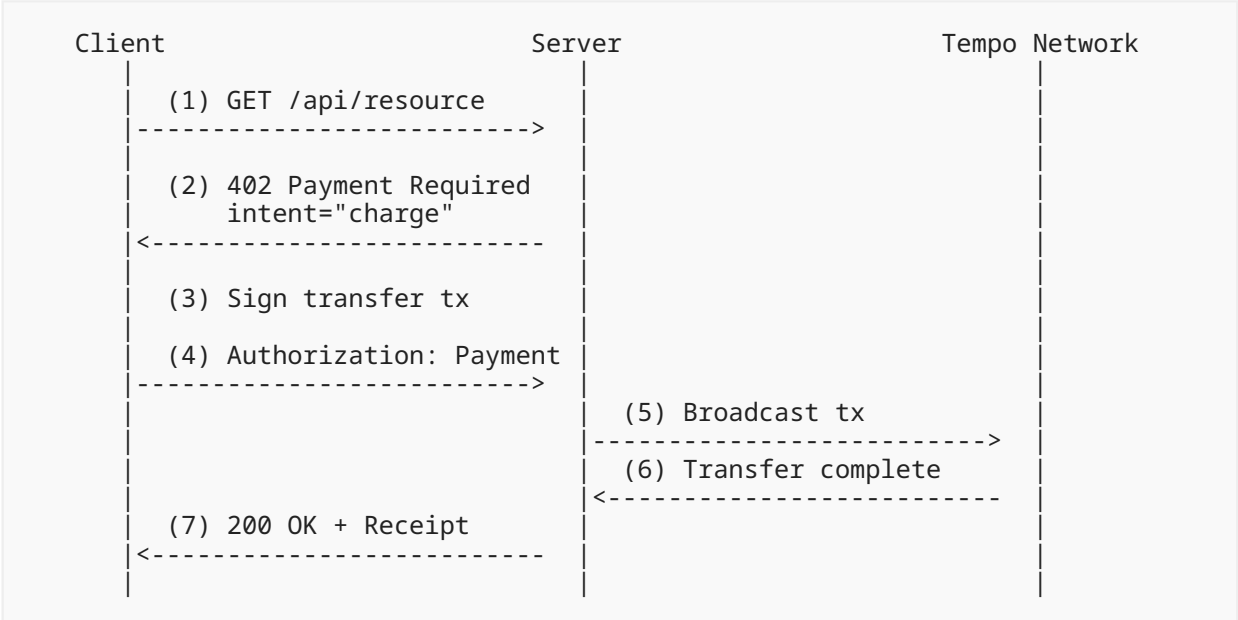
For non-zero charges, Tempo supports two submission modes:

- `pull`: The client signs a transaction and returns a `type="transaction"` credential for the server to broadcast.
- `push`: The client broadcasts the transaction and returns a `type="hash"` credential for the server to verify onchain.

Servers **SHOULD** support `pull` mode. Servers **MAY** additionally support `push` mode. Servers that do not support both non-zero modes for a challenge **MUST** advertise the supported subset via `methodDetails.supportedModes`.

1.1. Pull Mode (Default)

The default Tempo charge flow uses `pull` mode:



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

TIP-20 Tempo's enshrined token standard, implemented as precompiles rather than smart contracts. TIP-20 tokens use 6 decimal places and provide `transfer`, `transferWithMemo`, `transferFrom`, and `approve` operations.

Tempo Transaction An EIP-2718 transaction with type prefix `0x76`, supporting batched calls, multiple signature types (secp256k1, P256, WebAuthn), 2D nonces, and validity windows.

2D Nonce Tempo's nonce system where each account has multiple independent nonce lanes (`nonce_key`), enabling parallel transaction submission.

Fee Payer An account that pays transaction fees on behalf of another account. Tempo Transactions support fee payment via a separate signature domain (`0x78`), allowing the server to pay for fees while the client only signs the payment authorization.

4. Request Schema

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

4.1. Shared Fields

Field	Type	Required	Description
`amount`	string	**REQUIRED**	Amount in base units (stringified number)
`currency`	string	**REQUIRED**	TIP-20 token address (e.g., `"0x20c0..."`)
`recipient`	string	**REQUIRED**	Recipient address
`description`	string	**OPTIONAL**	Human-readable payment description
`externalId`	string	**OPTIONAL**	Merchant's 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	Required	Description
`methodDetails.chainId`	number	**OPTIONAL**	Tempo chain ID (default: 42431)
`methodDetails.feePayer`	boolean	**OPTIONAL**	If `true`, server pays transaction fees (default: `false`)

Field	Type	Required	Description
`methodDetails.memo`	string	**OPTIONAL**	A `bytes32` hex value. When present, the client **MUST** use `transferWithMemo` instead of `transfer`.
`methodDetails.splits`	array	**OPTIONAL**	Additional recipients that receive a portion of `amount`. See Section 4.4.
`methodDetails.supportedModes`	array	**OPTIONAL**	Supported non-zero submission modes. Values are `"pull"` and/or `"push"`.

Table 2

4.3. Submission Modes

The `supportedModes` field allows a server to advertise which non-zero charge submission modes it supports for a specific challenge.

- `pull` indicates that the client creates a `type="transaction"` credential containing a signed Tempo Transaction for the server to broadcast.
- `push` indicates that the client creates a `type="hash"` credential after broadcasting the transaction itself.

If `supportedModes` is present, it **MUST** contain at least one of `pull` or `push`, and clients **MUST** choose one of the advertised modes.

If `supportedModes` is omitted, clients **MAY** assume both `pull` and `push` are supported for that challenge for backwards compatibility with version 00 implementations. Servers **MUST** omit `supportedModes` only when they support both non-zero modes for the challenge. A server that supports only `pull` or only `push` for a challenge **MUST** include `supportedModes` explicitly and omit the unsupported mode.

For zero-amount charges, mode negotiation does not apply. Clients use a `type="proof"` credential regardless of `supportedModes`.

Example:

```
{
  "amount": "1000000",
  "currency": "0x20c0000000000000000000000000000000000000",
  "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00",
  "methodDetails": {
    "chainId": 4217,
    "feePayer": true,
    "supportedModes": ["pull"]
  }
}
```

The client fulfills this by signing a Tempo Transaction with `transfer(recipient, amount)` or `transferWithMemo(recipient, amount, memo)` on the specified `currency` (token address), with `validBefore` no later than the challenge `expires` auth-param. The client **MAY** use a dedicated `nonceKey` (2D nonce lane) for payment transactions to avoid blocking other account activity if the transaction is not immediately settled.

If `methodDetails.feePayer` is `true`, the client signs with `fee_payer_signature` set to `0x00` and `fee_token` empty, allowing the server to sponsor fees. If `feePayer` is `false` or omitted, the client **MUST** set `fee_token` and pay fees themselves.

4.4. Split Payments

The `splits` field enables a single charge to distribute payment across multiple recipients atomically. This is useful for platform fees, revenue sharing, and marketplace payouts.

4.4.1. Semantics

The top-level `amount` represents the total amount the client pays. Each entry in `splits` specifies a recipient and the amount they receive. The primary recipient (the top-level `recipient`) receives the remainder: `amount` minus the sum of all split amounts.

4.4.2. Split Entry Schema

Each entry in the `splits` array is a JSON object:

Field	Type	Required	Description
`amount`	string	**REQUIRED**	Amount in base units for this recipient
`memo`	string	**OPTIONAL**	A `bytes32` hex value for `transferWithMemo`
`recipient`	string	**REQUIRED**	Recipient address

Table 3

The `amount` field in each split entry **MUST** be a base-10 integer string with no sign, decimal point, exponent, or surrounding whitespace. Each `splits[i].amount` **MUST** be greater than zero. The syntax and encoding requirements for `splits[i].memo` are identical to those for `methodDetails.memo`, but apply only to that split transfer. Address fields are compared by decoded 20-byte value, not by string form.

4.4.3. Constraints

Servers **MUST NOT** generate a request where the sum of `splits[].amount` values is greater than or equal to `amount`. Clients **MUST** reject any request that violates this constraint. This ensures the primary recipient always receives a non-zero remainder, avoiding the need to define zero-value transfer semantics.

Additional constraints:

- If present, `splits` **MUST** contain at least 1 entry. Servers **SHOULD** limit splits to 10 entries to keep gas usage within a single block's budget (~29,000 gas per additional TIP-20 transfer). Servers **MAY** reject requests exceeding their supported split count.
- All transfers **MUST** target the same `currency` token address.

4.4.4. Ordering

The order of entries in `splits` is not significant for verification. Clients **SHOULD** emit calls in array order. Servers **MUST** verify that the required payment effects are present regardless of call ordering.

4.4.5. Example

```
{
  "amount": "1000000",
  "currency": "0x20c0000000000000000000000000000000000000",
  "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00",
  "methodDetails": {
    "chainId": 4217,
    "feePayer": true,
    "splits": [
      {
        "amount": "50000",
        "recipient": "0xA1B2C3D4E5F6A1B2C3D4E5F6A1B2C3D4E5F6A1B2"
      },
      {
        "amount": "10000",
        "memo":
"0x00000000000000000000000000000000000000000000000000000000deadbeef",
        "recipient": "0xC4D5E6F7A8B9C4D5E6F7A8B9C4D5E6F7A8B9C4D5"
      }
    ]
  }
}
```

This requests a total payment of 1.00 pathUSD (1,000,000 base units). The platform receives 0.05 pathUSD, the affiliate receives 0.01 pathUSD (with a memo), and the primary recipient receives the remaining 0.94 pathUSD (940,000 base units).

4.4.6. Client Behavior

When `splits` is present, the client **MUST** produce a transaction whose on-chain effects include the following `Transfer` or `TransferWithMemo` events on the `currency` token address:

1. The primary `recipient` receives `amount - sum(splits[].amount)`.
2. Each `splits[i].recipient` receives `splits[i].amount`. If `splits[i].memo` is present, the corresponding transfer **MUST** use `transferWithMemo`.

The top-level `methodDetails.memo`, if present, applies to the primary transfer.

Clients **MAY** achieve these effects using any valid transaction structure, including batched calls, smart contract wallet invocations, or intermediary operations such as token swaps — provided all required transfer events are emitted atomically.

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	Required	Description
`challenge`	object	**REQUIRED**	Echo of the challenge from the server
`payload`	object	**REQUIRED**	Tempo-specific payload object
`source`	string	**OPTIONAL**	Payer identifier as a DID (e.g., `did:pkh:eip155:4217:0x...`)

Table 4

The `source` field, if present, **SHOULD** use the `did:pkh` method with the chain ID applicable to the challenge and the payer's Ethereum address.

5.2. Transaction Payload (type="transaction")

When `type` is `"transaction"`, `signature` contains the complete signed Tempo Transaction (type 0x76) [TEMPO-TX-SPEC] serialized as RLP and hex-encoded with `0x` prefix. The transaction **MUST** authorize payment in the requested TIP-20 token sufficient to satisfy the challenge parameters, using one or more `transfer` and/or `transferWithMemo` calls. When `splits` are present, the transaction **MUST** include transfers for each split entry (see Section 4.4). This payload type corresponds to `pull` mode.

Field	Type	Required	Description
`signature`	string	**REQUIRED**	Hex-encoded RLP-serialized signed transaction
`type`	string	**REQUIRED**	`"transaction"`

Table 5

Example:

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "tempo",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "signature": "0x76f901...signed transaction bytes...",
    "type": "transaction"
  },
  "source": "did:pkh:eip155:4217:0x1234567890abcdef1234567890abcdef12345678"
}
```

5.3. Hash Payload (type="hash")

When `type` is `"hash"`, the client has already broadcast the transaction to the Tempo network. The `hash` field contains the transaction hash for the server to verify onchain. This payload type corresponds to `push` mode.

Field	Type	Required	Description
`hash`	string	**REQUIRED**	Transaction hash with `0x` prefix
`type`	string	**REQUIRED**	`"hash"`

Table 6

Example:

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "tempo",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "hash":
"0x1a2b3c4d5e6f7890abcdef1234567890abcdef1234567890abcdef1234567890",
    "type": "hash"
  },
  "source": "did:pkh:eip155:4217:0x1234567890abcdef1234567890abcdef12345678"
}
```

5.4. Proof Payload (type="proof")

When `amount` is `"0"`, no on-chain transfer is required. Instead of broadcasting a transaction, the client signs an EIP-712 typed-data message binding the proof to the challenge identifier. This payload type is used exclusively for zero-amount charges: clients **MUST** use `type="proof"` when `amount` is `"0"`, and **MUST NOT** use `type="proof"` when `amount` is non-zero. The `supportedModes` field does not apply to zero-amount charges.

Field	Type	Required	Description
`signature`	string	**REQUIRED**	EIP-712 signature with `0x` prefix
`type`	string	**REQUIRED**	`"proof"`

Table 7

The `source` field **MUST** be present on proof credentials and **MUST** be a `did:pkh:eip155:<chainId>:<address>` DID identifying the signer.

5.4.1. EIP-712 Domain and Types

The typed-data domain and types are:

```
{
  "domain": {
    "name": "MPP",
    "version": "3",
    "chainId": <challenge methodDetails.chainId>
  },
  "types": {
    "Proof": [
      { "name": "account", "type": "address" },
      { "name": "challengeId", "type": "string" },
      { "name": "realm", "type": "string" }
    ]
  },
  "primaryType": "Proof",
  "message": {
    "account": "<address from credential.source>",
    "challengeId": "<challenge.id>",
    "realm": "<challenge realm>"
  }
}
```

The `challengeId` in the message **MUST** be the `id` from the challenge that was issued to the client. This binds the signature to exactly one challenge, preventing cross-challenge replay. The `account` **MUST** be the address from `credential.source`, binding the proof to the paying account so that a signature cannot be replayed against another account controlled by the same signing key. The `realm` **MUST** be the `realm` from the challenge, binding the proof to the issuing origin and preventing cross-realm replay of a proof obtained from a different server.

5.4.2. Proof Contract Versions

The domain `version` identifies the proof contract. Three versions have shipped:

Version	Proof fields	Description
`"1"`	`challengeId`	Initial contract. Binds a proof to a single challenge.
`"2"`	`challengeId`, `realm`	Adds realm binding, so a proof obtained from one server cannot be replayed against another.
`"3"`	`account`, `challengeId`, `realm`	Adds payer binding, so a proof cannot be replayed against a different account, including across an access key authorized for multiple accounts.

Table 8

Implementations **MUST** use version `"3"`. The domain `version` is part of the EIP-712 digest, so a signature produced under version `"1"` or `"2"` does not verify against the version `"3"` typed data. Servers **MUST NOT** accept proof signatures made under domain version `"1"` or `"2"`.

5.4.3. Proof Verification

Servers **MUST** verify proof credentials as follows:

1. Verify `credential.source` is present and parses as `did:pkh:eip155:<chainId>:<address>`
2. Verify the chain ID from `source` matches `methodDetails.chainId` from the challenge
3. Reconstruct the typed data described above with `account` set to the address from `source`, `challengeId` set to the challenge `id`, and `realm` set to the challenge `realm`
4. Verify `payload.signature` is a valid signature over that typed data by the address from `source`. Recovering the signer and comparing it to the address from `source` satisfies this check; servers **MAY** additionally accept contract-account signatures (ERC-1271).
5. If step 4 fails, servers **MAY** instead recover the signer of `payload.signature` and accept the credential only if that signer is an access key currently authorized on-chain for the account in `source`. Servers **MUST NOT** otherwise accept a credential whose signer differs from the address in `source`.

5.4.4. Proof Receipt

Upon successful verification, servers return a receipt per [I-D.httpauth-payment] with `reference` set to the challenge `id` (since no on-chain transaction exists).

Example:

```
{
  "challenge": {
    "id": "kM9xPqWvT2nJrHsY4aDfEb",
    "realm": "api.example.com",
    "method": "tempo",
    "intent": "charge",
    "request": "eyJ...",
    "expires": "2025-02-05T12:05:00Z"
  },
  "payload": {
    "signature": "0xabcdef1234567890...",
    "type": "proof"
  },
  "source": "did:pkh:eip155:4217:0x1234567890abcdef1234567890abcdef12345678"
}
```

6. Fee Payment

When a request includes `feePayer: true`, the server commits to paying transaction fees on behalf of the client.

6.1. Server-Paid Fees

When `feePayer: true`:

1. **Client signs with placeholder**: The client signs the Tempo Transaction [TEMPO-TX-SPEC] with `fee_payer_signature` set to a placeholder value (`0x00`) and `fee_token` left empty. The client uses signature domain `0x76`.
2. **Server receives credential**: The server extracts the client-signed transaction from the credential payload.
3. **Server adds fee payment signature**: The server selects a `fee_token` (any USD-denominated TIP-20 stablecoin) and signs the transaction using signature domain `0x78`. This signature commits to the transaction including the `fee_token` and client's address.
4. **Server broadcasts**: The final transaction contains both signatures:
   - Client's signature (authorizing the payment)
   - Server's `fee_payer_signature` (committing to pay fees)

6.2. Client-Paid Fees

When `feePayer: false` or omitted, the client **MUST** set `fee_token` to a valid USD TIP-20 token address and pay fees themselves. The server broadcasts the transaction as-is without adding a fee payer signature.

6.3. Server Requirements

When acting as fee payer, servers:

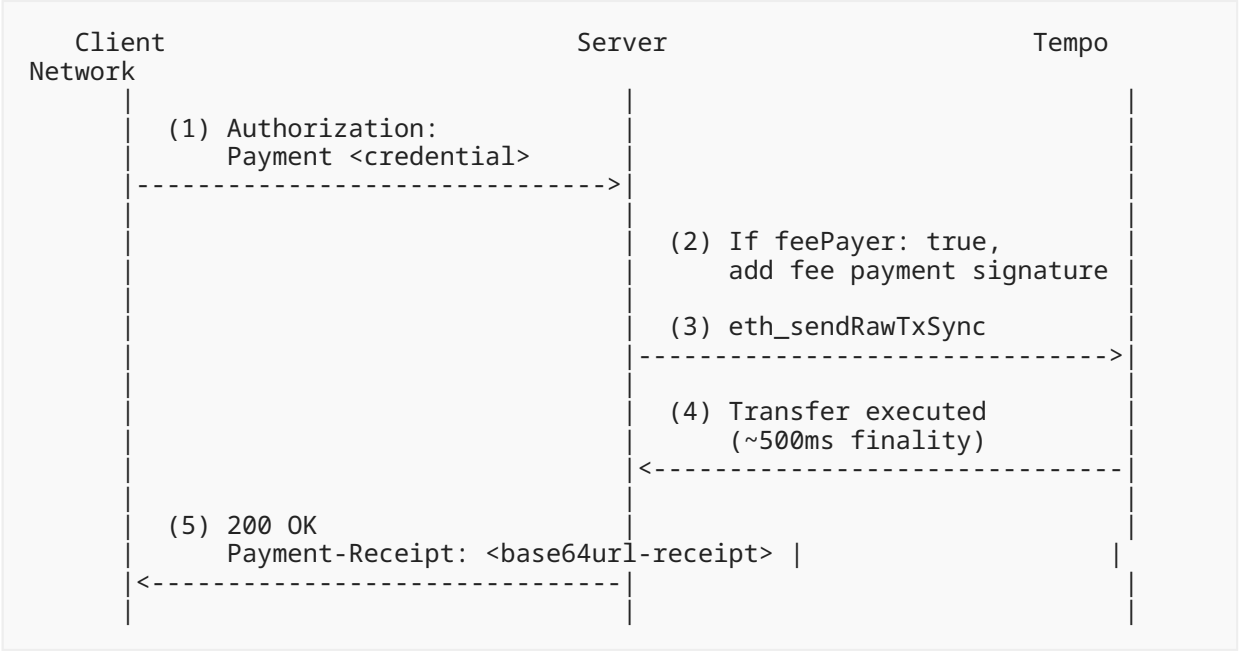
- **MUST** maintain sufficient balance of a USD TIP-20 token to pay transaction fees
- **MAY** use any USD-denominated TIP-20 token with sufficient AMM liquidity as the fee token
- **MAY** recover fee costs through pricing or other business logic

6.4. Client Requirements

- When `feePayer: true`: Clients **MUST** sign with `fee_payer_signature` set to `0x00` and `fee_token` empty or `0x80` (RLP null)
- When `feePayer: false` or omitted: Clients **MUST** set `fee_token` to a valid USD TIP-20 token and have sufficient balance to pay fees

7. Settlement Procedure

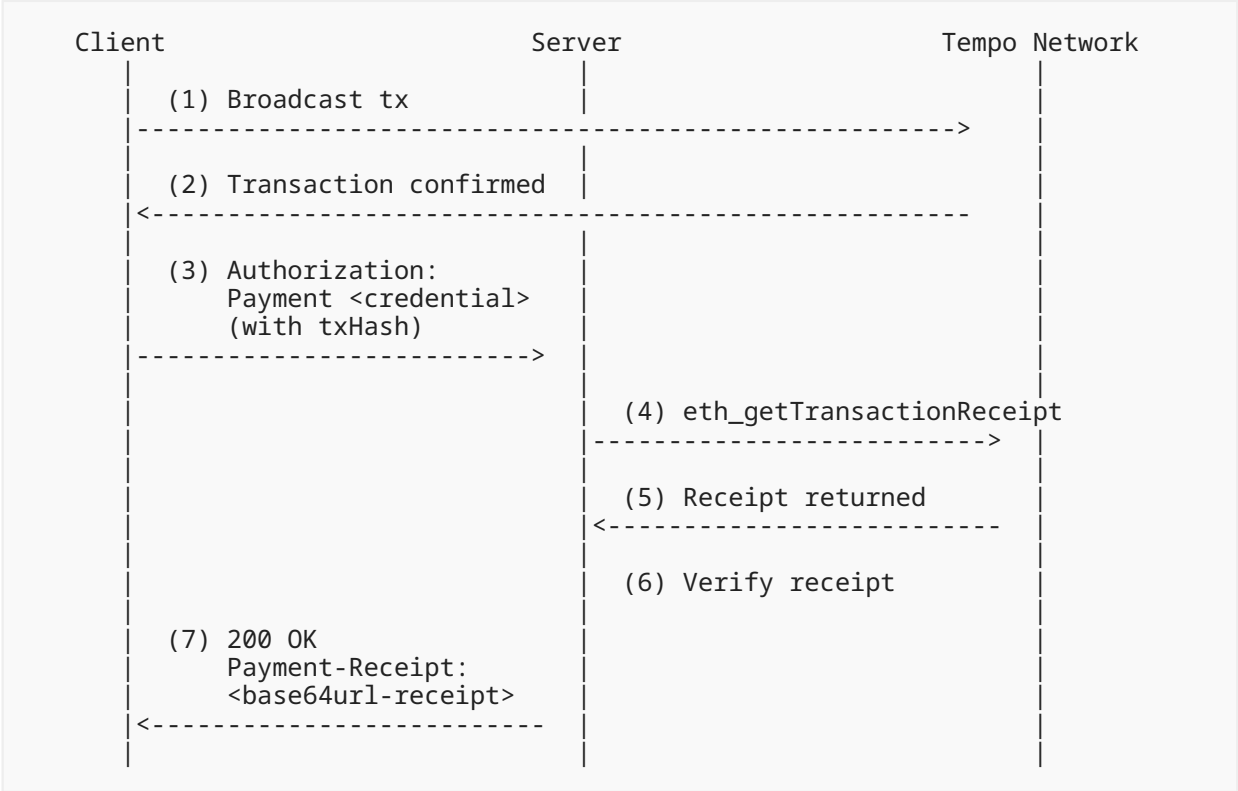
For `intent="charge"` fulfilled via transaction, the client signs a transaction containing one or more `transfer` or `transferWithMemo` calls. When `splits` are present, the transaction contains multiple calls (see Section 4.4). If `feePayer: true`, the server adds its fee payer signature before broadcasting:



1. Client submits credential containing signed `transfer` or `transferWithMemo` transaction
2. If `feePayer: true`, server adds fee sponsorship (signs with `0x78` domain)
3. Server broadcasts transaction to Tempo
4. Transaction included in block with immediate finality (~500ms)
5. Server returns a receipt whose `reference` field is the transaction digest

7.1. Hash Settlement

For credentials with `type="hash"`, the client has already broadcast the transaction. The server verifies the transaction onchain:



Limitations:

- Clients **MUST NOT** use `type="hash"` when `methodDetails.feePayer` is `true`. Servers **MUST** reject such credentials.
- If `methodDetails.supportedModes` is present and does not include `push`, clients **MUST NOT** use `type="hash"` credentials. Servers **MUST** reject such credentials.
- Server cannot modify or enhance the transaction.

7.2. Transaction Verification

Before broadcasting a transaction credential, servers **MUST** verify:

1. Deserialize the RLP-encoded transaction from `payload.signature`
2. Verify the transaction contains `transfer` or `transferWithMemo` calls on the `currency` token address
3. Verify the `amount` matches the challenge request amount
4. Verify the `recipient` matches the challenge request recipient
5. If `methodDetails.memo` is present, verify the transaction uses `transferWithMemo` with the matching memo value
6. If `methodDetails.splits` is present, verify the transaction includes transfers satisfying each split entry: the primary recipient receives `amount - sum(splits[].amount)`, each split recipient receives its specified amount, and any required memo values are present

7. If `methodDetails.supportedModes` is present, verify it includes `pull`

Servers **MAY** impose additional structural requirements (such as exact call count or ordering) as local policy before broadcasting.

7.3. Hash Verification

For hash credentials, servers **MUST** fetch the transaction receipt and verify that it indicates successful execution. Servers **MUST** verify that the receipt contains `Transfer` and/or `TransferWithMemo` event logs emitted by the `currency` token address whose payment effects satisfy the challenge parameters, including the primary recipient amount, any split amounts, and any required memo values.

If `methodDetails.supportedModes` is present, servers **MUST** verify it includes `push` before accepting a hash credential.

Servers **MAY** additionally inspect the transaction call data as a local-policy check, but call-data decoding is not required for conformance.

7.4. 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 Tempo charge:

Field	Type	Description
`method`	string	`"tempo"`
`reference`	string	Transaction hash of the settlement transaction
`status`	string	`"success"`
`timestamp`	string	[RFC3339] settlement time
`externalId`	string	**OPTIONAL.** Echoed from the challenge request

Table 9

8. Security Considerations

8.1. Transaction Replay

Tempo Transactions include chain ID, nonce, and optional `validBefore`/ `validAfter` timestamps that prevent replay attacks:

- Chain ID binding prevents cross-chain replay

- Nonce consumption prevents same-chain replay
- Validity windows limit temporal replay windows

8.2. Amount Verification

Clients **MUST** parse and verify the `request` payload before signing:

1. Verify `amount` is reasonable for the service
2. Verify `currency` is the expected token address
3. Verify `recipient` is controlled by the expected party
4. If `splits` is present, verify the sum of split amounts is strictly less than `amount` and that all split recipients are expected

8.3. Split Payment Risks

When `splits` are present, additional risks apply:

Recipient Transparency: Where a human approval step exists, clients **SHOULD** present each split recipient and amount so the user can verify the payment distribution. Clients **SHOULD** highlight when the primary recipient receives a small remainder relative to the total `amount`.

Gas Overhead: Each additional split adds approximately 29,000 gas for the TIP-20 precompile transfer execution. A charge with 10 splits adds approximately 290,000 gas beyond a single-transfer charge. Servers sponsoring fees via `feePayer: true` **MUST** budget for the increased gas limit.

Split Count Bound: Servers **SHOULD** limit `splits` to 10 entries. See Section 4.4 for rationale.

8.4. Server-Paid Fees

Servers acting as fee payers accept financial risk in exchange for providing a seamless payment experience.

Denial of Service: Malicious clients could submit valid-looking credentials that fail onchain, causing the server to pay fees without receiving payment. Servers **SHOULD** implement rate limiting and **MAY** require client authentication before accepting payment credentials.

Fee Token Exhaustion: Servers **MUST** monitor their fee token balance and reject new payment requests when balance is insufficient.

9. IANA Considerations

9.1. 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
`tempo`	Tempo blockchain TIP-20 token transfer	This document

Table 10

Contact: Tempo Labs (contact@tempo.xyz)

9.2. 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`	`tempo`	One-time TIP-20 transfer	This document

Table 11

10. References

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

10.2. Informative References

[EIP-2718] Zoltu, M., "Typed Transaction Envelope", October 2020, <https://eips.ethereum.org/EIPS/eip-2718>.

[EIP-55] Buterin, V., "Mixed-case checksum address encoding", January 2016, <https://eips.ethereum.org/EIPS/eip-55>.

[TEMPO-TX-SPEC] Tempo Labs, "Tempo Transaction Specification", n.d., <https://docs.tempo.xyz/protocol/transactions/spec-tempo-transaction>.

Appendix A. ABNF Collected

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

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

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

Appendix B. Example

Challenge:

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

request="eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiIweDIwYzAwMDAwMDAwMDAwMDAw
MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAiLCJyZWNpcGllbnQiOiIweDc0MmQzNUNjNjYzNEMwNTMyO
TI1YTNiODQ0QmM5ZTc1OTVmOGZFMDAiLCJtZXRob2REZXRhaWxzIjp7ImNoYWluSWQiOjQyMTcsIn
N1cHBvcnRlZE1vZGVzIjpbInB1bGwiXX19",
  expires="2025-01-06T12:00:00Z"
Cache-Control: no-store
```

The `request` decodes to:

```
{
  "amount": "1000000",
  "currency": "0x20c0000000000000000000000000000000000000",
  "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00",
  "methodDetails": {
    "chainId": 4217,
    "supportedModes": ["pull"]
  }
}
```

This requests a transfer of 1.00 pathUSD (1000000 base units).

Credential:

```
GET /api/resource HTTP/1.1
Host: api.example.com
Authorization: Payment
eyJjaGFsbGVuZ2UiOnsiaWQiOiJrTTl4UHFXdlQybkpySHNZNGFEZkViIn0sInBheWxvYWQiOnsic
2lnbmF0dXJlIjoiMHg3NmY5MDEuLi4iLCJ0eXBlIjoidHJhbnNhY3Rpb24ifSwic291cmNlIjoiZG
lkOnBraDplaXAxNTU6NDIxNzoweDEyMzQ1Njc4OTBhYmNkZWYxMjM0NTY3ODkwYWJjZGVmMTIzNDU
2NzgifQ
```

Appendix C. Split Payment Example

Challenge with splits:

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment id="sP1itPaym3ntEx4mple",
  realm="marketplace.example.com",
  method="tempo",
  intent="charge",

request="eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiIweDIwYzAwMDAwMDAwMDAwMDAw
MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAiLCJtZXRob2REZXRhaWxzIjp7ImNoYWluSWQiOjQyMTcsI
mZlZVBheWVyIjp0cnVlLCJzcGxpdHMiOlt7ImFtb3VudCI6IjUwMDAwIiwicmVjaXBpZW50IjoiMH
hBMUIyQzNENEU1RjZBMUIyQzNENEU1RjZBMUIyQzNENEU1RjZBMUIyIn1dfSwicmVjaXBpZW50Ijo
iMHg3NDJkMzVDYzY2MzRDMDUzMjkyNWEzYjg0NEJjOWU3NTk1ZjhmRTAwIn0",
  expires="2025-06-01T12:00:00Z"
Cache-Control: no-store
```

The `request` decodes to:

```
{
  "amount": "1000000",
  "currency": "0x20c0000000000000000000000000000000000000",
  "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00",
  "methodDetails": {
    "chainId": 4217,
    "feePayer": true,
    "splits": [
      {
        "amount": "50000",
        "recipient": "0xA1B2C3D4E5F6A1B2C3D4E5F6A1B2C3D4E5F6A1B2"
      }
    ]
  }
}
```

This requests a total payment of 1.00 pathUSD. The platform receives 0.05 pathUSD and the merchant receives 0.95 pathUSD. The resulting transaction must emit the following transfer events:

1. 950,000 to `0x742d...fE00` — merchant receives remainder
2. 50,000 to `0xA1B2...A1B2` — platform fee

Appendix D. Acknowledgements

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

Authors' Addresses

Jake Moxey
Tempo Labs
Email: jake@tempo.xyz

Brendan Ryan
Tempo Labs
Email: brendan@tempo.xyz

Tom Meagher
Tempo Labs
Email: tom@tempo.xyz