Workgroup:	Network Working Group
Internet-Draft:	draft-tempo-subscription-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 Subscription Intent for HTTP Payment Authentication

Abstract

This document defines the "subscription" intent for the "tempo" payment method in the Payment HTTP Authentication Scheme. It specifies how clients grant servers permission to collect a fixed TIP-20 token payment once per billing period using recipient-scoped access keys on the Tempo blockchain. This profile intentionally models the recurring transfer authorization itself, not a richer billing object.

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 `subscription` intent on Tempo represents a recurring fixed-amount TIP-20 payment. The client grants the server a recipient-scoped access key with a per-period spending limit. Activation registers the key and collects the first billing-period charge in the same transaction.

This specification inherits the shared `subscription` intent semantics from [I-D.payment-intent-subscription] and defines Tempo-specific request fields, payloads, and settlement behavior.

This profile is intentionally narrower than a general billing subscription. It standardizes a recurring token-transfer authorization, not price catalogs, quantities, prorations, deferred starts, or billing-anchor resets.

Tempo subscriptions support only key-authorization fulfillment. Tempo transactions containing standalone `approve` calls and push-mode hash credentials do not provide the per-period enforcement required for this intent.

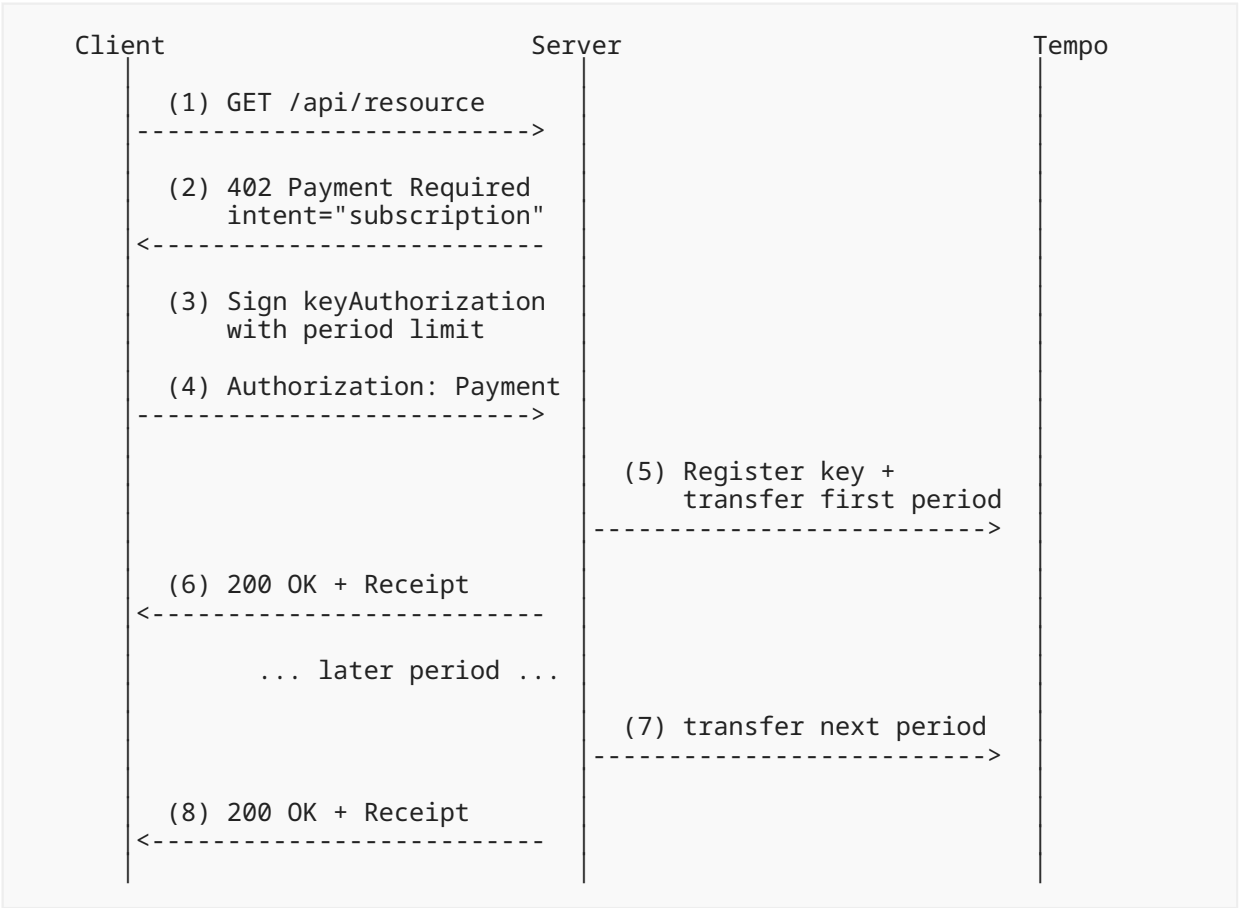
Tempo also imposes an additional constraint that is not part of the shared intent: the recurring authorization **MUST** have an explicit expiry. This method therefore requires a `subscriptionExpires` field because the underlying Tempo key authorization itself is time-bounded.

Tempo subscriptions also require the [TIP-1011] periodic token-limit and `allowed_calls` restrictions described in this document. Servers **MUST** reject request objects on chains or deployments that cannot enforce those restrictions.

The [TIP-1011] features required by this specification — periodic spending limits, `allowed_calls` target and selector scoping, and recipient-bound selector rules — are introduced in the Tempo T3 network upgrade. Servers **MUST NOT** issue `intent="subscription"` challenges on chains or deployments running a pre-T3 protocol version.

1.1. Subscription Flow

The following diagram illustrates the Tempo subscription flow:



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`, `transferFrom`, and `approve` operations.

Access Key A delegated signing key. For Tempo subscriptions, the access key is configured with an expiry timestamp, a per-period token spending limit, and a destination restriction.

AccountKeychain Precompile The Tempo precompile that manages access-key registration, spending limits, and periodic-limit enforcement [TEMPO-ACCOUNT-KEYCHAIN].

4. Request Schema

The `request` parameter in the `WWW-Authenticate` challenge contains a base64url-encoded JSON object. The `request` JSON **MUST** be serialized using JSON Canonicalization Scheme (JCS) [RFC8785] and base64url-encoded without padding per [I-D.httpauth-payment].

4.1. Request Fields

Tempo uses the shared `amount`, `currency`, `periodUnit`, `periodCount`, `subscriptionExpires`, `recipient`, `description`, and `externalId` fields from [I-D.payment-intent-subscription]. Tempo additionally requires `subscriptionExpires` because Tempo key authorizations must expire:

Field	Type	Required	Description
`amount`	string	**REQUIRED**	Fixed payment amount per billing period in base units
`currency`	string	**REQUIRED**	TIP-20 token address
`periodUnit`	string	**REQUIRED**	Billing period unit. The value **MUST** be `day` or `week`
`periodCount`	string	**REQUIRED**	Positive integer count of `periodUnit` values per billing period
`subscriptionExpires`	string	**REQUIRED**	Subscription expiry timestamp in [RFC3339] format
`recipient`	string	**REQUIRED**	Recipient address authorized for subscription charges
`description`	string	**OPTIONAL**	Human-readable subscription description
`externalId`	string	**OPTIONAL**	Merchant's reference for the subscription

Table 1

The `amount` value **MUST** be a string representation of a positive integer in base 10 with no sign, decimal point, exponent, or surrounding whitespace. Leading zeros **MUST NOT** be used.

The `periodCount` value **MUST** be a string representation of a positive integer in base 10 with no sign, decimal point, exponent, or surrounding whitespace. Leading zeros **MUST NOT** be used.

Hex values in this profile use lowercase hexadecimal with `0x` prefix and no padding or truncation. Implementations **MUST** use lowercase hex when generating addresses, token identifiers, selectors, and hex-encoded signed payloads. Implementations **SHOULD** accept mixed-case input, but **MUST** normalize it to lowercase before comparison. Address, selector, and token-identifier comparisons are by decoded value, not raw string form.

4.2. Method Details

Field	Type	Required	Description
`methodDetails.accessKey`	object	**REQUIRED**	Access key descriptor that the payer authorizes for subscription renewal charges
`methodDetails.accessKey.accessKeyAddress`	string	**REQUIRED**	Address of the access key to authorize
`methodDetails.accessKey.keyType`	string	**REQUIRED**	Access key type. The value **MUST** be `p256`, `secp256k1`, or `webAuthn`
`methodDetails.chainId`	number	**OPTIONAL**	Tempo chain ID. If omitted, the default value is 42431 (Tempo mainnet).

Table 2

Servers issuing `intent="subscription"` challenges **SHOULD** include the `expires` auth-param in `WWW-Authenticate` per [I-D.httpauth-payment], using [RFC3339] format. Request objects **MUST NOT** duplicate the challenge expiry value. The `subscriptionExpires` field instead defines when the subscription itself expires.

If the challenge includes `expires`, the `subscriptionExpires` value **MUST** be strictly later than the challenge `expires` timestamp. Servers **MUST** reject credentials where `subscriptionExpires` is at or before the challenge `expires`.

Tempo subscriptions map the shared period fields to the [TIP-1011] `TokenLimit` `period` field as follows:

- `periodUnit="day"` maps to `periodCount * 86400` seconds

- `periodUnit="week"` maps to `periodCount * 604800` seconds

Servers **MUST** reject `periodUnit="month"` because [TIP-1011] periodic token limits are fixed elapsed-time periods and cannot represent calendar-month billing exactly. Servers **MUST** reject request objects where the mapped period cannot be represented as an unsigned 64-bit integer. Servers **MUST** reject request objects where `subscriptionExpires` cannot be represented in the Tempo key authorization expiry field.

Example:

```
{
  "amount": "10000000",
  "currency": "0x20c0000000000000000000000000000000000001",
  "periodUnit": "day",
  "periodCount": "30",
  "subscriptionExpires": "2026-07-14T12:00:00Z",
  "recipient": "0x742d35cc6634c0532925a3b844bc9e7595f8fe00",
  "methodDetails": {
    "accessKey": {
      "accessKeyAddress": "0x1111111111111111111111111111111111111111",
      "keyType": "p256"
    },
    "chainId": 42431
  }
}
```

The client fulfills this by signing a key authorization with:

- Expiry = `subscriptionExpires`
- Access key = `methodDetails.accessKey`
- Per-period spending limit = `amount`
- Billing period = mapped period in seconds
- Destination restriction = `recipient`

The signed key authorization **MUST** additionally configure:

- the exact access-key address and key type from `methodDetails.accessKey`
- a `TokenLimit` for `currency` whose `amount` equals the challenge `amount` and whose `period` equals the mapped period in seconds
- exactly one `allowed_calls` target scope whose `target` equals `currency`
- explicit selector rules for `transfer(address,uint256)` (`0xa9059cbb`) and optionally `transferWithMemo(address,uint256,bytes32)` (`0x95777d59`)
- a recipient allowlist for each permitted selector containing only the challenge `recipient`

The signed key authorization **MUST NOT** use unrestricted target mode for the subscription token, and it **MUST NOT** authorize `approve` or any other non-transfer selector.

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:eip155:42431:0x...`)

Table 3

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. Key Authorization Payload (type="keyAuthorization")

Subscriptions on Tempo **MUST** use `type="keyAuthorization"`. The `signature` field contains the complete signed key authorization serialized as RLP and hex-encoded with `0x` prefix.

The encoded value **MUST** be a signed key authorization containing at least:

- the Tempo chain ID
- the access-key address and key type
- the authorization expiry
- the TIP-20 token spending limit
- the billing-period limit configuration
- the recipient restriction
- the `allowed_calls` scope described above

The embedded signature **MUST** use a primitive signature type supported by [TIP-1020]. Keychain wrapper signatures **MUST NOT** be used for this field.

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

Table 4

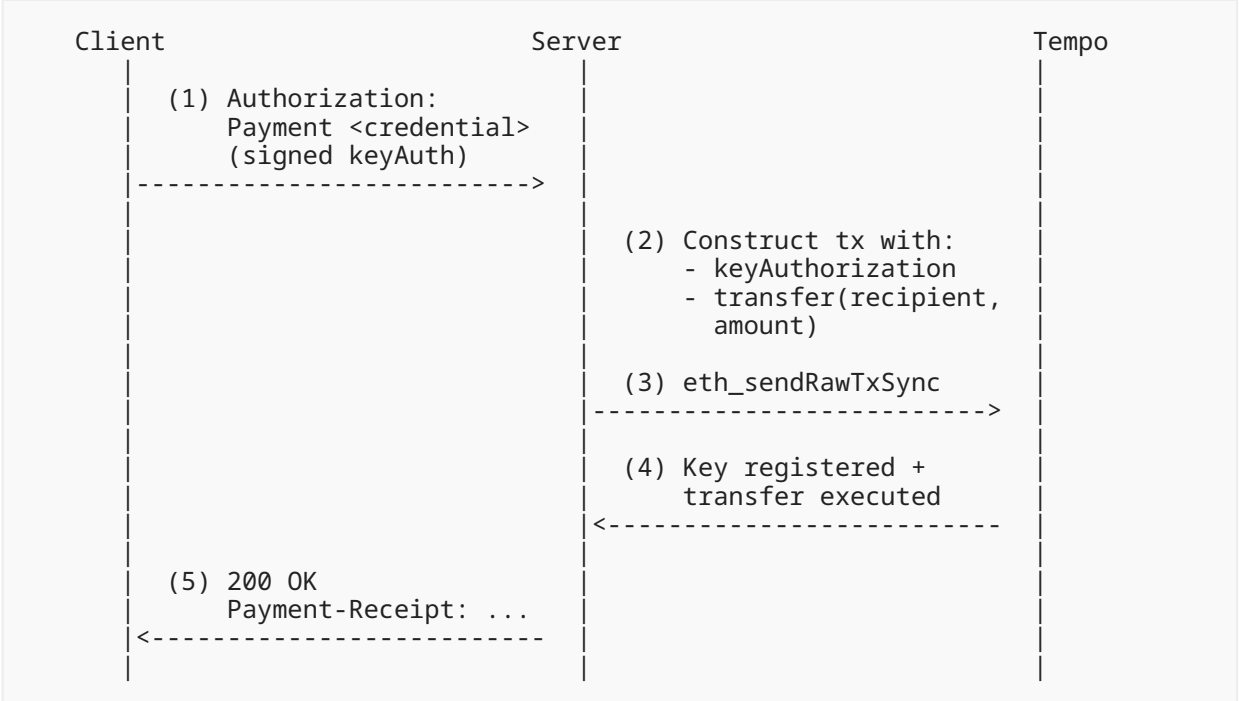
Example:

```
{
  "challenge": {
    "id": "qT8wErYuI3oPlKjH6gFdSa",
    "realm": "api.example.com",
    "method": "tempo",
    "intent": "subscription",
    "request": "eyJ...",
    "expires": "2026-01-15T12:05:00Z"
  },
  "payload": {
    "signature": "0xf8c1...signed authorization bytes...",
    "type": "keyAuthorization"
  },
  "source": "did:pkh:eip155:42431:0x1234567890abcdef1234567890abcdef12345678"
}
```

6. Settlement Procedure

6.1. Activation and First-Period Charge

For `intent="subscription"`, activation and the first billing-period charge are a single atomic operation:



Servers **MUST** treat the subscription as active only after the activation transaction succeeds.

Servers **MUST NOT** treat activation as successful if the activation transaction settles at or after `subscriptionExpires`.

6.2. Renewal

For each later billing period, the server **MAY** submit one transaction using the registered access key to transfer `amount` to `recipient`.

Servers **MUST NOT** submit more than one successful renewal charge for the same billing period.

6.3. Billing Anchor and Subscription State

The billing anchor for a Tempo subscription is the block timestamp, or equivalent consensus settlement timestamp, of the block containing the successful activation transaction. Servers **MUST** derive this anchor from chain settlement data rather than local wall-clock time.

Billing periods are defined as:

- Period 0: `[anchor, anchor + mappedPeriodSeconds)`
- Period 1: `[anchor + mappedPeriodSeconds, anchor + 2*mappedPeriodSeconds)`
- Period N: `[anchor + N*mappedPeriodSeconds, anchor + (N+1)*mappedPeriodSeconds)`

Servers **MUST** maintain durable local state for each subscription, including at least:

- subscription identifier
- billing anchor
- last charged billing-period index
- any in-flight billing-period index and renewal transaction identifier
- subscription expiry
- revocation status

When granting access in a later billing period, servers **MUST**:

- Verify the subscription has not expired or been revoked
- Determine the current billing-period index from the anchor and the mapped period in seconds
- Verify that the current billing period has not already been charged
- Atomically record any renewal attempt for the current billing period as in-flight before submitting the renewal transaction
- Mark the current billing period as charged only after the renewal transaction settles successfully
- Grant access only after, or atomically with, durably recording the successful renewal charge

For duplicate idempotent requests, servers **MUST NOT** charge the same billing period more than once.

[TIP-1011] periodic spending limits reset to one billing period of capacity and do not accumulate across elapsed periods. If one or more billing periods elapse without a successful renewal charge, a later transaction authorizes at most one charge in the then-current billing period. Servers **MUST NOT** treat missed billing periods as additional on-chain spending capacity.

6.4. Source Verification

If a credential includes the optional `source` field, servers **MUST NOT** trust this value without verification.

Servers **MUST** verify the payer identity by recovering the root signer address from the signed key authorization using [TIP-1020]-compatible verification semantics over the encoded key authorization payload.

If `source` is present, servers **MUST** verify that it identifies the recovered root signer on the same chain as `methodDetails.chainId`, or on chain 42431 when `methodDetails.chainId` is omitted.

6.5. Authorization Scope Verification

When validating a Tempo subscription credential, servers **MUST** verify that the signed key authorization expiry equals `subscriptionExpires`. Servers **MUST** verify that the signed key authorization chain ID equals `methodDetails.chainId`, or 42431 when `methodDetails.chainId` is omitted. Servers **MUST** verify that the signed key authorization authorizes the exact access key described by `methodDetails.accessKey`, including both `accessKeyAddress` and `keyType`. Servers **MUST** also verify that the authorization contains a spending limit for `currency` whose amount equals `amount` and whose billing period equals the mapped period in seconds.

Servers **MUST** verify that the signed key authorization's `allowed_calls` scope:

- contains exactly one target scope, and that scope is for `currency`
- uses explicit selector rules rather than unrestricted target mode
- allows `transfer(address,uint256)` and **MAY** additionally allow `transferWithMemo(address,uint256,bytes32)`
- does not allow `approve(address,uint256)` or any other non-transfer selector
- restricts the first ABI `address` argument for each permitted selector to the challenge `recipient`

Servers **MUST** reject authorizations that permit spending the subscription token through broader call scopes than those required above.

6.6. Receipt Generation

Upon successful activation or renewal, servers **MUST** return a `Payment-Receipt` header per [I-D.httpauth-payment]. Servers **MUST NOT** include a `Payment-Receipt` header on error responses.

On activation, servers **MUST** include the `subscriptionId` defined by [I-D.payment-intent-subscription] in the receipt. On renewal, servers **MUST** return the same `subscriptionId` for the active subscription.

The receipt payload for Tempo subscription:

Field	Type	Description
`method`	string	`"tempo"`
`reference`	string	Transaction hash of the settlement transaction
`status`	string	`"success"`
`subscriptionId`	string	Server-issued opaque identifier for the subscription
`timestamp`	string	[RFC3339] settlement time
`externalId`	string	**OPTIONAL**. Echoed from the challenge request

Table 5

7. Security Considerations

7.1. Destination Scoping

Tempo subscription access keys **MUST** be restricted to the `recipient` address in the request. Where [TIP-1011] recipient-bound selector rules are available, servers **MUST** reject credentials that do not enforce this restriction through `allowed_calls`.

7.2. Amount and Period 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 TIP-20 token address
3. Verify `periodUnit` and `periodCount` match expectations
4. Verify `recipient` is controlled by the expected party
5. Verify `subscriptionExpires` is acceptable

7.3. Revocation

Users can revoke subscription access keys at any time via the AccountKeychain precompile [TEMPO-ACCOUNT-KEYCHAIN]. Servers **SHOULD** handle revocation gracefully by returning a fresh subscription challenge.

7.4. Duplicate Charge Prevention

On-chain periodic limits prevent overspending within a billing period, but they do not by themselves make HTTP service delivery idempotent [TEMPO-ACCOUNT-KEYCHAIN]. Servers **MUST** implement durable local state to prevent duplicate renewal charges caused by retries or concurrent requests.

7.5. Key Scope Minimization

Subscription access keys **SHOULD** use the narrowest [TIP-1011] scope needed to support recurring charges. Implementations **SHOULD** avoid unrestricted target scopes and **SHOULD** limit the key to the subscription token, the permitted transfer selectors, and the configured recipient.

7.6. Access Key Isolation

Servers **SHOULD** generate a unique key pair and use a distinct access key for each subscription. This provides fault isolation: compromise of one server-held key affects only the subscription associated with that key, and revoking one subscription's key via `revokeKey()` does not invalidate other active subscriptions between the same payer and server.

If a server reuses a single access key across multiple subscriptions from the same payer, the key's permissions must be broad enough to cover all active subscriptions — potentially spanning multiple tokens, recipients, or spending limits. This widens the blast radius if the key is compromised and forces revocation of all subscriptions at once. It also complicates spending-limit accounting, since [TIP-1011] enforces a single `TokenLimit` per `(account, key, token)` tuple: two subscriptions for the same token on the same key would share one periodic limit rather than being independently capped.

Servers that reuse keys across subscriptions **MUST** ensure the combined `TokenLimit` and `allowed_calls` scope still satisfies the per- subscription authorization scope verification requirements in this document. In practice this is difficult to guarantee, and implementations **SHOULD** prefer one key per subscription for simplicity and security.

7.7. Caching

Responses to subscription challenges (402 Payment Required) **MUST** include `Cache-Control: no-store` to prevent sensitive payment data from being cached by intermediaries.

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

8. IANA Considerations

The `subscription` payment intent is registered by [I-D.payment-intent-subscription]. This document does not register it again.

9. 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/>.

[I-D.payment-intent-subscription] Moxey, J., "Subscription Intent for HTTP Payment Authentication", April 2026, <https://datatracker.ietf.org/doc/draft-payment-intent-subscription/>.

[TEMPO-ACCOUNT-KEYCHAIN] Tempo Labs, "Account Keychain Precompile", n.d., <https://docs.tempo.xyz/protocol/precompiles/account-keychain>.

[TIP-1011] Goyal, T., "TIP-1011: Enhanced Access Key Permissions", n.d., <https://docs.tempo.xyz/protocol/tips/tip-1011>.

[TIP-1020] Tempo Labs, "TIP-1020: Signature Verification Precompile", n.d., <https://docs.tempo.xyz/protocol/tips/tip-1020>.

Appendix A. Examples

This section is non-normative.

A.1. Activation

Challenge:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="qT8wErYuI3oPlKjH6gFdSa",
  realm="api.example.com",
  method="tempo",
  intent="subscription",
  expires="2026-01-15T12:05:00Z",
  request="<base64url-encoded JSON below>"
```

The `request` decodes to:

```
{
  "amount": "10000000",
  "currency": "0x20c0000000000000000000000000000000000001",
  "periodUnit": "day",
  "periodCount": "30",
  "subscriptionExpires": "2026-07-14T12:00:00Z",
  "recipient": "0x742d35cc6634c0532925a3b844bc9e7595f8fe00",
  "methodDetails": {
    "accessKey": {
      "accessKeyAddress": "0x1111111111111111111111111111111111111111",
      "keyType": "p256"
    },
    "chainId": 42431
  }
}
```

This requests a recurring payment of 10.00 alphaUSD every 30 days until 2026-07-14T12:00:00Z.

Credential:

```
{
  "challenge": {
    "id": "qT8wErYuI3oPlKjH6gFdSa",
    "realm": "api.example.com",
    "method": "tempo",
    "intent": "subscription",
    "request": "eyJ...",
    "expires": "2026-01-15T12:05:00Z"
  },
  "payload": {
    "signature": "0xf8c1...signed authorization bytes...",
    "type": "keyAuthorization"
  },
  "source": "did:pkh:eip155:42431:0x1234567890abcdef1234567890abcdef12345678"
}
```

The activation transaction submitted by the server contains:

- the signed `keyAuthorization`

- one TIP-20 `transfer(recipient, amount)` call, or `transferWithMemo(recipient, amount, memo)` if the implementation uses a memo

If activation settles at `2026-01-15T12:03:10Z`, the `Payment-Receipt` payload decodes to:

```
{
  "method": "tempo",
  "reference":
"0x8d7c6c0d94d8488cb4cf6ab7b8a2f9c3f8e0eac7e5b6d1e8c3d86f733c2b7c01",
  "status": "success",
  "subscriptionId": "c3ViXzAxMjM0NTY",
  "timestamp": "2026-01-15T12:03:10Z"
}
```

The server records at least:

- `subscriptionId = "c3ViXzAxMjM0NTY"`
- `billing anchor = 2026-01-15T12:03:10Z`
- `periodUnit = "day"`
- `periodCount = "30"`
- `mappedPeriodSeconds = 2592000`
- `accessKeyAddress = "0x1111111111111111111111111111111111111111"`
- `last charged billing-period index = 0`

A.2. Renewal Across Multiple Periods

Using the activation timestamp above, the Tempo subscription billing periods are:

- Period 0: `[2026-01-15T12:03:10Z, 2026-02-14T12:03:10Z)`
- Period 1: `[2026-02-14T12:03:10Z, 2026-03-16T12:03:10Z)`
- Period 2: `[2026-03-16T12:03:10Z, 2026-04-15T12:03:10Z)`

Requests during Period 0 can use the active subscription without a new authorization:

```
GET /api/resource HTTP/1.1
Host: api.example.com
Cookie: session=<application-session>
```

When Period 1 begins, the server determines that billing-period index 1 has not yet been charged. The server submits one Tempo transaction using the registered access key to call the TIP-20 token at `currency` with:

- `transfer(recipient, amount)`, or
- `transferWithMemo(recipient, amount, memo)`

If that transaction settles successfully, the renewal `Payment-Receipt` payload decodes to:

```
{
  "method": "tempo",
  "reference":
"0xb4bf2b4f8e3f0e6f3b6af3a5f6d3c8e32e1c32a19fa56bd9f9b3fd33af88e912",
  "status": "success",
  "subscriptionId": "c3ViXzAxMjM0NTY",
  "timestamp": "2026-02-14T12:05:42Z"
}
```

The server updates `last charged billing-period index = 1`. Additional requests during Period 1 do not permit another successful renewal charge for Period 1.

A.3. Cancellation At Period End

Suppose the server has already successfully charged Period 2 and the payer cancels on `2026-03-20T09:00:00Z`.

The server records cancellation with an effective time of `2026-04-15T12:03:10Z`, which is the end of Period 2. Requests before that time continue to succeed without another renewal charge:

```
GET /api/resource HTTP/1.1
Host: api.example.com
Cookie: session=<application-session>
```

Once `2026-04-15T12:03:10Z` is reached, the server stops submitting renewal transactions for this subscription. A later request receives a fresh challenge:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="n3xtP3ri0d",
  realm="api.example.com",
  method="tempo",
  intent="subscription",
  request="<base64url-encoded JSON below>"
```

A.4. Failed Renewal And Lapse

Suppose Period 3 begins and the server attempts a renewal transaction, but the Tempo transaction fails because the payer no longer has enough TIP-20 balance or fee-paying balance.

The server does not grant access for Period 3 and returns:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="r3tryP3ri0d",
  realm="api.example.com",
  method="tempo",
  intent="subscription",
  request="<base64url-encoded JSON below>"
```

If a later retry during Period 3 succeeds, the server may grant access for Period 3 and update `last charged billing-period index = 3`.

If Period 4 begins before any successful renewal occurs, the next successful Tempo transaction authorizes at most one charge for Period 4. The elapsed unpaid Period 3 does not become extra on-chain spending capacity, because [TIP-1011] periodic limits reset rather than accumulate.

A.5. Natural Expiry

Suppose `subscriptionExpires` is `2026-07-14T12:00:00Z`. Once that time is reached, the signed key authorization no longer authorizes future renewals. Requests after that time receive a fresh challenge rather than another renewal attempt.

Appendix B. Acknowledgements

The authors thank the MPP 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: thomas@tempo.xyz