

---

Workgroup: Network Working Group  
Internet-Draft: draft-payment-transport-mcp-00  
Published: 18 May 2026  
Intended Status: Informational  
Expires: 19 November 2026  
Authors: J. Moxey B. Ryan  
*Tempo Labs Tempo Labs*

# Payment Authentication Scheme: JSON-RPC & MCP Transport

---

## Abstract

This document defines how the Payment HTTP Authentication Scheme operates over JSON-RPC 2.0 transports. It specifies the mapping of payment challenges to JSON-RPC error responses using implementation-defined error codes, credential transmission via metadata fields, receipt delivery in successful responses, and error handling conventions. This specification applies to any transport carrying JSON-RPC 2.0 messages, including WebSocket, HTTP, stdio, and protocol frameworks such as the Model Context Protocol (MCP).

## 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. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

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	3
1.1. Applicability	4
1.2. Design Goals	4
2. Requirements Language	4
3. Terminology	5
4. Protocol Overview	5
5. Capability Advertisement	5
5.1. MCP Capability Advertisement	6
6. Payment Challenge	7
6.1. Signaling Payment Required	7
6.2. Challenge Structure	8
6.3. Multiple Payment Options	9
7. Payment Credential	10
7.1. Metadata Placement	10
7.2. Transmitting Payment Data	11
7.3. Credential Structure	12
8. Payment Receipt	12
8.1. Successful Payment Response	12
8.2. Receipt Structure	13
9. MCP Covered Operations	13
9.1. Tool Calls	13
9.2. Resource Access	14
9.3. Prompt Retrieval	15

---

10. Error Handling	15
10.1. Error Code Mapping	15
10.2. Payment Verification Failure	16
10.3. Protocol Errors	17
11. Notifications	17
11.1. Server Behavior	17
11.2. Client Guidance	17
12. Security Considerations	18
12.1. Challenge Binding	18
12.2. Replay Protection	18
12.3. Transport Security	18
12.4. Credential Confidentiality	18
12.5. Metadata Stripping	19
12.6. Confused Deputy	19
12.7. Denial of Service	19
13. IANA Considerations	19
14. Normative References	19
Appendix A. Example: Ethereum JSON-RPC over WebSocket	20
Appendix B. Example: MCP Tool Call	23
Appendix C. References to MCP Specification	25
Authors' Addresses	25

## 1. Introduction

JSON-RPC 2.0 [JSON-RPC] is a stateless, lightweight remote procedure call protocol using JSON [RFC8259]. Many modern protocols layer JSON-RPC over various transports including HTTP, WebSocket [RFC6455], and stdio. Protocol frameworks such as the Model Context Protocol (MCP) [MCP] also use JSON-RPC 2.0 as their message format. This document defines how the Payment HTTP Authentication Scheme [I-D.httpauth-payment] operates within JSON-RPC 2.0 messages, independent of the underlying transport.

This specification defines:

- Error codes for payment signaling
- Challenge structure in JSON-RPC error responses
- Credential transmission via `_meta` metadata fields
- Receipt delivery via `_meta` metadata fields
- Error handling conventions
- Notification behavior
- Capability advertisement

## 1.1. Applicability

This specification applies to any system that exchanges JSON-RPC 2.0 messages, including but not limited to:

- **WebSocket:** JSON-RPC over persistent `wss://` connections for real-time APIs, streaming services, and subscriptions.
- **HTTP:** JSON-RPC 2.0 over standard HTTP request-response exchanges.
- **stdio:** JSON-RPC 2.0 over standard input/output streams for local process communication.

This specification also defines MCP-specific conventions for the Model Context Protocol [MCP], which uses JSON-RPC 2.0 for tool invocations (`tools/call`), resource access (`resources/read`), and prompt retrieval (`prompts/get`).

Transport-specific security requirements (e.g., TLS for WebSocket, process isolation for stdio) are addressed in [Section 12.3](#).

## 1.2. Design Goals

1. **Native JSON:** Use JSON objects directly rather than base64url encoding, leveraging JSON-RPC's native capabilities.
2. **Transport Independent:** Define payment semantics at the JSON-RPC layer, applicable to any transport carrying JSON-RPC messages.
3. **Minimal Overhead:** Add payment data only when needed via the `_meta` extension mechanism.
4. **Multiple Options:** Support servers offering multiple payment methods in a single challenge.

## 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

This document uses terminology from [\[I-D.httpauth-payment\]](#):

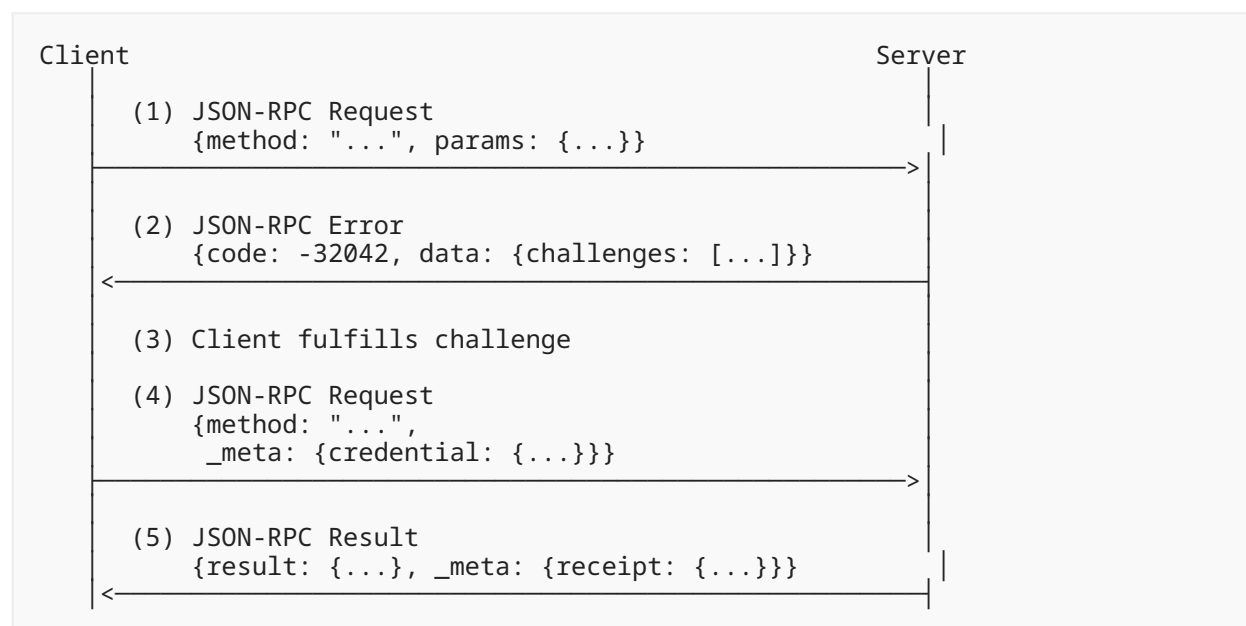
**Challenge** Payment requirements communicated by the server.

**Credential** Payment authorization data sent by the client.

**Receipt** Server acknowledgment of successful payment.

### 4. Protocol Overview

The payment flow follows three phases within JSON-RPC message exchanges:



### 5. Capability Advertisement

Servers and clients **SHOULD** advertise supported payment methods and intents before payment flows begin. The capability object **SHOULD** contain:

**methods (REQUIRED):** Object mapping payment method identifiers (as registered in the IANA HTTP Payment Methods registry) to their configuration. Each method object **MUST** contain an **intents** array listing the supported payment intent types (as registered in the IANA HTTP Payment Intents registry) for that method.

Example capability object:

```
{
  "methods": {
    "tempo": { "intents": ["charge"] },
    "stripe": { "intents": ["charge"] }
  }
}
```

The mechanism for advertising capabilities depends on the transport:

- **WebSocket:** Servers **MAY** send a `payment.capabilities` JSON-RPC notification after connection establishment.
- **MCP:** See [Section 5.1](#).

Clients **MAY** use capability information to determine compatibility before invoking paid methods. Clients **MUST NOT** rely solely on capability advertisement to determine payment support; malicious servers could claim capabilities they don't properly implement. Clients **SHOULD** validate challenge structure before fulfilling payment.

## 5.1. MCP Capability Advertisement

For MCP specifically, servers **SHOULD** advertise payment support in the `InitializeResult`:

```
{
  "protocolVersion": "2025-11-25",
  "capabilities": {
    "tools": {},
    "resources": {},
    "experimental": {
      "payment": {
        "methods": {
          "tempo": { "intents": ["charge"] },
          "stripe": { "intents": ["charge"] }
        }
      }
    }
  },
  "serverInfo": {
    "name": "example-server",
    "version": "1.0.0"
  }
}
```

Clients **SHOULD** advertise in the `InitializeRequest`:

```
{
  "protocolVersion": "2025-11-25",
  "capabilities": {
    "experimental": {
      "payment": {
        "methods": {
          "tempo": { "intents": ["charge"] }
        }
      }
    }
  },
  "clientInfo": {
    "name": "example-client",
    "version": "1.0.0"
  }
}
```

Servers **MAY** use client capabilities to filter which payment options to offer in challenges.

## 6. Payment Challenge

### 6.1. Signaling Payment Required

When a JSON-RPC method requires payment, the server **MUST** respond with a JSON-RPC error using code -32042 (Payment Required). This code is within the JSON-RPC implementation-defined server error range (-32000 to -32099) per [\[JSON-RPC\]](#):

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [
        {
          "id": "qB3wErTyU7iOpAsD9fGhJk",
          "realm": "api.example.com",
          "method": "tempo",
          "intent": "charge",
          "request": {
            "amount": "1000",
            "currency": "usd",
            "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
          },
          "expires": "2025-01-15T12:05:00Z",
          "description": "API call fee"
        }
      ],
      "problem": {
        "type": "https://paymentauth.org/problems/payment-required",
        "title": "Payment Required",
        "status": 402,
        "detail": "Payment required for access."
      }
    }
  }
}
```

The `error.data.httpStatus` field **SHOULD** be included with value `402` to indicate the corresponding HTTP status code for transports that bridge to HTTP (e.g., MCP Streamable HTTP).

## 6.2. Challenge Structure

The `error.data` object **MUST** contain:

**challenges** (REQUIRED): Array of one or more challenge objects.

**problem** (OPTIONAL): An RFC 9457 Problem Details object providing additional error context. When present, contains `type`, `title`, `status`, `detail`, and optionally `challengeId`.

Each challenge object **MUST** contain:

**id** (REQUIRED): Unique challenge identifier. Servers **MUST** cryptographically bind this value to at minimum the following parameters: `realm`, `method`, `intent`, `request` (canonical hash), and `expires`. Clients **MUST** include this value unchanged in the credential.

**realm** (REQUIRED): Protection space identifier defining the scope of the payment requirement.

**method (REQUIRED):** Payment method identifier as registered in the IANA HTTP Payment Methods registry.

**intent (REQUIRED):** Payment intent type as registered in the IANA HTTP Payment Intents registry.

**request (REQUIRED):** Method-specific payment request data as a native JSON object. Servers **MUST NOT** base64url-encode the request when using JSON-RPC transport. For challenge binding and challenge ID verification, both parties **MUST** canonicalize request using JSON Canonicalization Scheme (JCS) [\[RFC8785\]](#) and hash the canonicalized bytes. The schema is defined by the payment method specification.

Each challenge object **MAY** contain:

**expires (OPTIONAL):** Timestamp in [\[RFC3339\]](#) format after which the challenge is no longer valid. Clients **SHOULD NOT** attempt to fulfill challenges past their expiry. If absent, servers define the validity period.

**description (OPTIONAL):** Human-readable description of what the payment is for.

### 6.3. Multiple Payment Options

When multiple challenges are present, they represent **alternative** payment options. Clients **MUST** select exactly one challenge to fulfill. Servers **MUST NOT** require multiple simultaneous payments via this mechanism.

Servers **MAY** offer multiple payment options by including multiple challenge objects in the challenges array:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [
        {
          "id": "pT7yHnKmQ2wErXsZ5vCbNl",
          "realm": "api.example.com",
          "method": "tempo",
          "intent": "charge",
          "request": {
            "amount": "1000",
            "currency": "usd"
          }
        },
        {
          "id": "mF8uJkLp03qRtYsA6wDcVb",
          "realm": "api.example.com",
          "method": "stripe",
          "intent": "charge",
          "request": {
            "amount": "1000",
            "currency": "usd"
          }
        }
      ]
    }
  }
}
```

Clients **SHOULD** select one challenge based on their capabilities and user preferences. Clients **MUST** send only one credential corresponding to a single selected challenge.

## 7. Payment Credential

### 7.1. Metadata Placement

This specification defines two placement strategies for the `_meta` field, depending on the protocol:

**Root-level `_meta`** (Generic JSON-RPC): The `_meta` field is placed at the root of the JSON-RPC message object. This approach works with any JSON-RPC method regardless of whether `params` is an object or an array:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "eth_getBlockByNumber",
  "params": ["latest", false],
  "_meta": {
    "org.paymentauth/credential": { ... }
  }
}
```

**Nested `_meta`** (MCP): The `_meta` field is placed inside `params` (for requests) or `result` (for responses), per MCP conventions where `params` is always an object:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "expensive-api",
    "_meta": {
      "org.paymentauth/credential": { ... }
    }
  }
}
```

Servers **MUST** check both locations for `_meta` and **MUST NOT** require clients to use a specific placement. Servers **MUST** ignore `org.paymentauth/credential` on methods that do not require payment.

## 7.2. Transmitting Payment Data

Clients send payment credentials using the `_meta` field with key `org.paymentauth/credential`. The key uses reverse-DNS naming to avoid collisions with other extensions:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "eth_getBlockByNumber",
  "params": ["latest", false],
  "_meta": {
    "org.paymentauth/credential": {
      "challenge": {
        "id": "qB3wErTyU7iOpAsD9fGhJk",
        "realm": "api.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {
          "amount": "1000",
          "currency": "usd",
          "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
        }
      },
      "expires": "2025-01-15T12:05:00Z"
    },
    "payload": {
      "signature": "0x1b2c3d4e5f..."
    }
  }
}
```

### 7.3. Credential Structure

The `org.paymentauth/credential` object **MUST** contain:

**challenge (REQUIRED):** The complete challenge object from the server's -32042 error response. Clients **MUST** echo the challenge unchanged.

**payload (REQUIRED):** Method-specific payment proof as a JSON object. The schema is defined by the payment method specification.

The credential object **MAY** contain:

**source (OPTIONAL):** Identifier of the payment source (e.g., a DID or address).

## 8. Payment Receipt

### 8.1. Successful Payment Response

After successful payment verification and settlement, servers **MUST** include a receipt using `_meta` with key `org.paymentauth/receipt`. The `_meta` field placement follows the same rules as [Section 7.1](#): root-level for generic JSON-RPC, nested in `result` for MCP.

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "number": "0x1348c9",
    "hash": "0x7736fab79e05dc611604d22470dadad2..."
  },
  "_meta": {
    "org.paymentauth/receipt": {
      "status": "success",
      "method": "tempo",
      "timestamp": "2025-01-15T12:00:30Z",
      "reference": "tx_abc123...",
      "challengeId": "qB3wErTyU7i0pAsD9fGhJk"
    }
  }
}
```

Servers **MUST** return `org.paymentauth/receipt` on every successful response to a paid request. Servers **MUST NOT** return receipts for unpaid requests.

## 8.2. Receipt Structure

The `org.paymentauth/receipt` object **MUST** contain:

**status** (REQUIRED): Settlement status. **MUST** be "success" for successful payments.

**method** (REQUIRED): Payment method that was used.

**timestamp** (REQUIRED): [RFC3339] timestamp of settlement.

**challengeId** (REQUIRED): The id from the fulfilled challenge.

The receipt object **MAY** contain:

**reference** (OPTIONAL): Method-specific settlement reference (e.g., transaction hash, invoice ID).

## 9. MCP Covered Operations

The following MCP operations support payment flows.

### 9.1. Tool Calls

Tool invocations via `tools/call` **MAY** require payment:

**Challenge:**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "premium-analysis"
  }
}
```

**Response:**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "tool-pay-123",
        "realm": "tools.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {"amount": "500", "currency": "usd"}
      }]
    }
  }
}
```

## 9.2. Resource Access

Resource reads via `resources/read` **MAY** require payment:

**Challenge:**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "resources/read",
  "params": {
    "uri": "data://premium/market-data"
  }
}
```

**Response:**

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "resource-pay-456",
        "realm": "data.example.com",
        "method": "stripe",
        "intent": "charge",
        "request": {"amount": "100", "currency": "usd"}
      }]
    }
  }
}
```

### 9.3. Prompt Retrieval

Prompt retrieval via prompts/get **MAY** require payment:

#### Response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "prompt-pay-789",
        "realm": "prompts.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {"amount": "50", "currency": "usd"}
      }]
    }
  }
}
```

## 10. Error Handling

### 10.1. Error Code Mapping

Servers **MUST** map payment errors to JSON-RPC error codes within the server error range (-32000 to -32099) per [\[JSON-RPC\]](#):

Condition	Code	Description
Payment required	-32042	Payment challenge in error . data
Payment verification failed	-32043	Fresh challenge + failure reason
Malformed credential	-32602	Invalid params (bad JSON structure)
Internal payment error	-32603	Payment processor failure

Table 1

Note: -32700 (Parse error) applies only when the entire JSON-RPC message is unparseable, not for malformed `_meta` subfields. Use -32602 for credential structure errors.

## 10.2. Payment Verification Failure

When payment verification fails, servers **MUST** return code -32043 with a fresh challenge and failure details:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "error": {
    "code": -32043,
    "message": "Payment Verification Failed",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "retry-challenge-abc",
        "realm": "api.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {"amount": "1000", "currency": "usd"}
      }],
      "failure": {
        "reason": "signature-invalid",
        "detail": "Signature verification failed"
      }
    }
  }
}
```

On verification failure, servers **MAY** return the same challenge if it remains valid, or issue a fresh challenge. Clients **SHOULD** treat any -32043 response as requiring a new payment attempt.

The failure object **MAY** contain:

**reason** (OPTIONAL): Machine-readable failure code.

**detail** (OPTIONAL): Human-readable failure description.

### 10.3. Protocol Errors

Technical errors use standard JSON-RPC error codes:

#### Invalid Params (-32602):

Used for malformed credentials or missing required fields:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
      "detail": "Missing required field: challenge.id"
    }
  }
}
```

## 11. Notifications

JSON-RPC notifications are requests without an `id` field that expect no response. Since payment challenges require a response, notifications cannot support payment flows.

### 11.1. Server Behavior

Servers **MUST NOT** process payment-gated operations invoked as notifications. Servers **SHOULD** silently drop such notifications per JSON-RPC 2.0 semantics.

Servers **MAY** log dropped payment-required notifications for debugging purposes.

Servers **MAY** send JSON-RPC notifications to deliver data after a client has fulfilled a prior payment challenge (e.g., streaming subscription updates over WebSocket).

### 11.2. Client Guidance

Clients **SHOULD NOT** invoke payment-gated operations as notifications. Operations that may require payment **SHOULD** always include a request `id` to receive payment challenges and results.

## 12. Security Considerations

### 12.1. Challenge Binding

Servers **MUST** cryptographically bind challenge IDs to their parameters (at minimum: `realm`, `method`, `intent`, `request hash`, `expires`). The request hash **MUST** be computed over the JCS-canonicalized representation of request per [RFC8785]. This prevents clients from reusing a challenge ID with modified payment terms.

Servers **SHOULD** also bind challenges to the specific operation being requested (e.g., tool name, resource URI) to prevent a challenge issued for one operation being used for another.

### 12.2. Replay Protection

Servers **MUST** reject credentials for:

- Unknown challenge IDs
- Expired challenges (past `expires` timestamp)
- Previously-used challenge IDs

Servers **SHOULD** maintain a record of used challenge IDs for at least the challenge validity period. For high-throughput scenarios, servers **MAY** use stateless challenge tokens (e.g., MAC over canonical parameters) and maintain only a post-use replay set.

When two concurrent requests race using the same challenge, servers **MUST** ensure only one succeeds via atomic check-and-mark operations.

### 12.3. Transport Security

When using network transports (HTTP, WebSocket), all communication **MUST** occur over TLS 1.2 [RFC5246] or later (TLS 1.3 [RFC8446] **RECOMMENDED**).

For stdio transport, security depends on the process isolation provided by the operating system.

For persistent transports (e.g., WebSocket), servers **SHOULD** additionally rate limit connection establishment and close connections that exceed challenge request thresholds.

### 12.4. Credential Confidentiality

Payment credentials **MAY** contain sensitive data (signatures, tokens). Clients **MUST NOT** log or persist credentials beyond immediate use. Servers **MUST NOT** log full credential payloads. This includes crash dumps, distributed tracing, and analytics telemetry.

## 12.5. Metadata Stripping

On-path attackers or malicious intermediaries could strip `org.paymentauth/credential` from requests (causing repeated payment challenges) or `org.paymentauth/receipt` from responses (affecting auditability). For network transports, TLS provides integrity. For stdio transport, rely on process isolation.

## 12.6. Confused Deputy

Clients may be tricked into paying for unintended operations if method names or realms are misleading. Client implementations **SHOULD**:

- Display realm, amount, currency, and recipient to users before fulfilling payment challenges
- Allow users to configure payment policies per realm
- Validate that challenge parameters match the requested operation

## 12.7. Denial of Service

Attackers may trigger many payment challenges to exhaust server resources or payment processor rate limits. Servers **SHOULD**:

- Rate limit challenge issuance per client
- Use stateless challenge encoding where possible
- Implement exponential backoff for repeated failures

## 13. IANA Considerations

This document has no IANA actions. Payment methods and intents are registered per [[I-D.httpauth-payment](#)].

Servers **MUST** use the following JSON-RPC error codes:

Code	Name	Description
-32042	Payment Required	Server requires payment to proceed
-32043	Payment Verification Failed	Payment credential invalid

*Table 2*

These codes are within the JSON-RPC server error range (-32000 to -32099). Implementations **MUST** use these exact codes for interoperability.

## 14. 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>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <<https://www.rfc-editor.org/info/rfc8785>>.
- [I-D.httpauth-payment] Moxey, J., "The 'Payment' HTTP Authentication Scheme", January 2026, <<https://datatracker.ietf.org/doc/draft-ryan-httpauth-payment/>>.
- [MCP] "Model Context Protocol Specification", n.d., <<https://modelcontextprotocol.io/specification/2025-11-25>>.
- [JSON-RPC] "JSON-RPC 2.0 Specification", n.d., <<https://www.jsonrpc.org/specification>>.

## Appendix A. Example: Ethereum JSON-RPC over WebSocket

An `eth_getBlockByNumber` call over WebSocket with payment required for RPC access:

### Connection:

```
GET / HTTP/1.1
Host: rpc.example.com
Upgrade: websocket
Connection: Upgrade
```

### Step 1: Initial Request

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "eth_getBlockByNumber",
  "params": ["latest", false]
}
```

### Step 2: Payment Challenge

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "ch_ws_789",
        "realm": "rpc.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {
          "amount": "1",
          "currency": "usd",
          "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
        }
      }],
      "expires": "2025-01-15T12:05:00Z",
      "description": "Ethereum RPC call"
    }
  }
}
```

### Step 3: Request with Credential

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "eth_getBlockByNumber",
  "params": ["latest", false],
  "_meta": {
    "org.paymentauth/credential": {
      "challenge": {
        "id": "ch_ws_789",
        "realm": "rpc.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {
          "amount": "1",
          "currency": "usd",
          "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
        }
      },
      "expires": "2025-01-15T12:05:00Z"
    },
    "source": "0x1234567890abcdef...",
    "payload": {
      "signature": "0xabc123..."
    }
  }
}
```

#### Step 4: Success with Receipt

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "number": "0x1348c9",
    "hash": "0x7736fab79e05dc611604d22470dadad2...",
    "parentHash": "0x61cdb2a09ab99abf791d474f20c2ea...",
    "timestamp": "0x56ffeff8",
    "gasLimit": "0x47e7c4",
    "gasUsed": "0x38658",
    "miner": "0xf8b483dba2c3b7176a3da549ad41a48b...",
    "transactions": []
  },
  "_meta": {
    "org.paymentauth/receipt": {
      "status": "success",
      "method": "tempo",
      "timestamp": "2025-01-15T12:00:15Z",
      "reference": "0xtx789...",
      "challengeId": "ch_ws_789"
    }
  }
}
```

## Appendix B. Example: MCP Tool Call

A complete MCP tool call with payment, using nested `_meta` per MCP conventions:

### Step 1: Initial Request

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "web-search",
    "arguments": {"query": "MCP protocol"}
  }
}
```

### Step 2: Payment Challenge

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -32042,
    "message": "Payment Required",
    "data": {
      "httpStatus": 402,
      "challenges": [{
        "id": "ch_abc123",
        "realm": "search.example.com",
        "method": "tempo",
        "intent": "charge",
        "request": {
          "amount": "10",
          "currency": "usd",
          "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
        }
      },
      "expires": "2025-01-15T12:05:00Z",
      "description": "Web search query"
    }
  ]
}
}
```

### Step 3: Request with Payment

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "web-search",
    "arguments": {"query": "MCP protocol"},
    "_meta": {
      "org.paymentauth/credential": {
        "challenge": {
          "id": "ch_abc123",
          "realm": "search.example.com",
          "method": "tempo",
          "intent": "charge",
          "request": {
            "amount": "10",
            "currency": "usd",
            "recipient": "0x742d35Cc6634C0532925a3b844Bc9e7595f8fE00"
          },
          "expires": "2025-01-15T12:05:00Z"
        },
        "source": "0x1234567890abcdef...",
        "payload": {
          "signature": "0xabc123..."
        }
      }
    }
  }
}
```

#### Step 4: Success with Receipt

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "content": [{
      "type": "text",
      "text": "Search results for 'MCP protocol'..."
    }],
    "_meta": {
      "org.paymentauth/receipt": {
        "status": "success",
        "method": "tempo",
        "timestamp": "2025-01-15T12:00:15Z",
        "reference": "0xtx789...",
        "challengeId": "ch_abc123"
      }
    }
  }
}
```

## Appendix C. References to MCP Specification

- MCP Base Protocol: <https://modelcontextprotocol.io/specification/2025-11-25/basic>
- MCP Transports: <https://modelcontextprotocol.io/specification/2025-11-25/basic/transports>
- MCP Tools: <https://modelcontextprotocol.io/specification/2025-11-25/server/tools>
- MCP Resources: <https://modelcontextprotocol.io/specification/2025-11-25/server/resources>
- MCP `_meta` Field: <https://modelcontextprotocol.io/specification/2025-11-25/basic>

## Authors' Addresses

**Jake Moxey**

Tempo Labs

Email: [jake@tempo.xyz](mailto:jake@tempo.xyz)**Brendan Ryan**

Tempo Labs

Email: [brendan@tempo.xyz](mailto:brendan@tempo.xyz)