# MoltsPay: A Protocol for Autonomous Agent-to-Agent Payments

Zen7 Labs
`support@moltspay.com`

March 2026

**Abstract**

As AI agents become increasingly autonomous, they require the ability to transact economically without human intervention. This paper introduces MoltsPay, an open-source protocol and SDK enabling AI agents to hold funds, make payments, and receive payments using cryptocurrency. Built on the x402 HTTP payment standard and leveraging gasless transaction infrastructure, MoltsPay provides a practical solution for agent-to-agent commerce. We describe the protocol architecture, security model, and reference implementations in Python and TypeScript.

## Introduction

### The Economic Agent Problem

The rapid advancement of large language models (LLMs) and autonomous AI agents has created a new class of software entities capable of complex, goal-directed behavior. These agents can browse the web, write code, schedule meetings, and perform multi-step tasks with minimal human oversight.

However, current AI agents face a fundamental limitation: **they cannot participate in economic transactions**. When an agent encounters a paid API, a premium data source, or another agent offering services for a fee, it must either:

1. Fail the task entirely
2. Request human intervention to complete payment
3. Rely on pre-negotiated API keys with fixed quotas

None of these options support true autonomous operation. As agents become more capable and are deployed at scale, this economic limitation becomes a critical bottleneck.

### Requirements for Agent Payments

An effective agent payment system must satisfy several requirements:

- **Programmable**: Payments must be executable via code without human interaction
- **Permissionless**: Agents should not require bank accounts, credit checks, or identity verification
- **Low-latency**: Transactions must complete in seconds, not days
- **Low-cost**: Fees must be negligible for micro-transactions
- **Secure**: Spending must be bounded and auditable
- **Interoperable**: The system must work across different agent frameworks and platforms

### Contribution

This paper presents MoltsPay, a protocol satisfying all of the above requirements. Our contributions include:

1. A protocol specification for HTTP-native agent payments based on the x402 standard
2. A gasless transaction architecture eliminating the need for agents to hold native blockchain tokens
3. A spending limit mechanism providing cryptographic guarantees on maximum expenditure
4. Open-source reference implementations in Python and TypeScript
5. A service discovery mechanism enabling agents to find and evaluate paid services

## Background

### HTTP 402 Payment Required

The HTTP 402 status code was reserved in the original HTTP/1.1 specification (RFC 2616) for "future use" in digital payment systems. The x402 protocol (https://x402.org) provides a modern implementation of this concept, making payments a native part of the HTTP request/response cycle.

The flow operates as follows:

1. Client sends request to server
2. Server returns 402 Payment Required with payment details

3. Client signs payment and retries request
4. Server verifies payment on-chain and delivers service

### Stablecoins and Layer 2 Networks

Cryptocurrency volatility makes it unsuitable for everyday transactions. Stablecoins—tokens pegged to fiat currencies—solve this problem. MoltsPay supports three major stablecoins:

- **USDC** (USD Coin) - Regulated reserves, 1:1 USD peg, issued by Circle
- **USDT** (Tether) - Largest stablecoin by market cap, widely accepted
- **DAI** (MakerDAO) - Decentralized stablecoin option

Layer 2 (L2) networks provide scalability improvements over Ethereum mainnet:

Table 1: Supported networks and their characteristics

| Network | Chain ID | Avg. Fee | Confirmation Time |
|---|---|---|---|
| Ethereum L1 | 1 | $2-50 | 12 seconds |
| Base | 8453 | $0.001-0.01 | 2 seconds |
| Polygon | 137 | $0.01-0.05 | 2 seconds |
| Base Sepolia (testnet) | 84532 | Free | 2 seconds |

### Gasless Transactions

Traditional blockchain transactions require the sender to hold native tokens (ETH) to pay for gas. This creates friction for AI agents, which would need to manage two token balances.

ERC-4337 (Account Abstraction) and paymaster infrastructure enable "gasless" transactions where a third party sponsors gas fees. MoltsPay leverages Coinbase's paymaster service, allowing agents to transact using only stablecoins.

## Protocol Specification

### Service Discovery

MoltsPay services advertise their capabilities via a well-known endpoint. The configuration follows the official JSON Schema at https://moltspay.com/schemas/services.json.

```
{
  "provider": {
    "name": "Zen7 Video Generation",
    "wallet": "0xb8d6f2441e8f8dfB6288A74Cf73804cDd0484E0C",
```

```json
    "chain": "base",
    "chains": ["base", "base_sepolia", "polygon"]
  },
  "services": [
    {
      "id": "text-to-video",
      "name": "Text to Video",
      "function": "textToVideo",
      "price": 0.99,
      "currency": "USDC",
      "input": {
        "prompt": {"type": "string", "required": true}
      }
    }
  ]
}
```

## Payment Flow

The complete payment flow consists of four phases:

**Phase 1: Discovery** - Client fetches `/.well-known/agent-services.json`

**Phase 2: Request** - Client sends service request

**Phase 3: Payment Challenge** - Server returns 402 with payment requirements

**Phase 4: Signed Request** - Client signs payment and retries

## Signature Scheme

MoltsPay uses EIP-712 typed structured data signing with chain-aware configuration. This provides:

- Multi-chain support across Base, Polygon, and testnets
- Multi-token support (USDC, USDT, DAI)
- Replay protection via nonce
- Time-bounded validity via deadline

## Transaction Settlement

MoltsPay supports two settlement modes:

**Immediate Settlement (Default):** The server submits the transaction immediately upon receiving a valid signature. The request blocks until on-chain confirmation ( 2 seconds on Base).

**Optimistic Settlement:** For trusted clients or low-value transactions, the server may return results immediately after signature verification, settling the transaction asynchronously.

## Security Model

### Spending Limits

MoltsPay enforces spending limits at the wallet level:

```
client.init_wallet(
    max_per_tx=10.0,       # Maximum per transaction
    max_per_day=100.0      # Maximum per 24-hour period
)
```

These limits are stored in the wallet configuration and enforced client-side before signing. For stronger guarantees, agents can use smart contract wallets with on-chain spending limits.

### Threat Model

Table 2: Threat model and mitigations

| Threat | Mitigation |
|---|---|
| Malicious service overcharging | Client verifies price before signing |
| Replay attacks | Nonce included in signature |
| Man-in-the-middle | HTTPS required; address verified |
| Wallet key compromise | Spending limits bound maximum loss |
| Service non-delivery | On-chain receipt enables disputes |

## Implementation

Reference implementations are available in Python and TypeScript:

```
from moltspay import MoltsPay

client = MoltsPay()
client.init_wallet(max_per_tx=10.0, max_per_day=100.0)

result = client.pay(
    service_url="https://api.zen7.com",
    service_id="text-to-video",
    prompt="A serene Japanese garden"
)
```

### Framework Integrations

MoltsPay provides native integrations for LangChain, CrewAI, and other popular agent frameworks.

## Economics

### Fee Structure

MoltsPay itself charges no protocol fees. Total cost per transaction is less than $0.01, consisting primarily of network gas fees.

### Market Dynamics

Agent-to-agent payments enable new market structures:

- **Specialization**: Agents focus on core competencies, outsourcing other tasks
- **Price discovery**: Competitive markets emerge for common services
- **Reputation systems**: On-chain payment history enables trust scoring

## Related Work

Prior work on agent payments includes the Autonomous Economic Agents (AEA) framework by Fetch.ai, SingularityNET marketplace, and Ocean Protocol. MoltsPay differentiates by focusing on simplicity, HTTP-native integration, and compatibility with mainstream agent frameworks.

## Future Work

- Multi-party payments for composite services
- Streaming payments for continuous services
- Cross-chain settlement via bridges
- Decentralized service registry with reputation

## Conclusion

MoltsPay provides a practical solution for AI agent payments, combining the programmability of cryptocurrency with the simplicity of HTTP APIs. By eliminating gas fees and providing built-in spending limits, we enable safe autonomous agent commerce.

The protocol is fully open-source and available at:

- Python SDK: https://pypi.org/project/moltspay/
- TypeScript SDK: https://www.npmjs.com/package/moltspay
- GitHub: https://github.com/Yaqing2023/moltspay-python
- Schema: https://moltspay.com/schemas/services.json

## References

1. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1
2. EIP-712 - Typed structured data hashing and signing
3. EIP-4337 - Account Abstraction Using Alt Mempool
4. x402 Protocol Specification - https://x402.org
5. MoltsPay Services Schema - https://moltspay.com/schemas/services.json
6. USDC Technical Documentation - https://developers.circle.com
7. Base Network Documentation - https://docs.base.org