# MoltsPay: Universal Payment Protocol for AI Agents

Zen7 Labs

March 2026

## Contents

**MoltsPay: Universal Payment Protocol for AI Agents** **2**
Abstract . . . 2
1. Introduction . . . 2
1.1 The Problem . . . 2
1.2 Our Contribution: Universal Payment Protocol . . . 2
2. Universal Payment Protocol Architecture . . . 2
2.1 Protocol Abstraction Layer . . . 2
2.2 Protocol Selection Logic . . . 4
2.3 Unified Payment Flow . . . 4
3. Supported Protocols . . . 4
3.1 x402 Protocol (Base, Polygon) . . . 4
3.2 MPP - Machine Payment Protocol (Tempo) . . . 4
3.3 Solana Pay-for-Success Protocol . . . 4
3.4 BNB Pre-Approval Protocol . . . 5
4. Multi-Chain Support . . . 5
4.1 Supported Networks . . . 5
4.2 Cross-VM Unification . . . 5
4.3 Server Multi-Chain Configuration . . . 6
5. Facilitator Architecture . . . 6
5.1 Facilitator Registry . . . 6
5.2 External vs Self-Hosted Facilitators . . . 6
5.3 Facilitator Interface . . . 7
6. Security Model . . . 7
6.1 Client Security . . . 7
6.2 Server Security . . . 7
6.3 Protocol Security . . . 7
7. Implementation . . . 7
7.1 SDK Support . . . 7
7.2 Client Usage . . . 8
7.3 Server Usage . . . 8
8. Future Work . . . 8
8.1 Additional Protocols . . . 8
8.2 Protocol Improvements . . . 8
8.3 Decentralization . . . 8
9. Conclusion . . . 8
References . . . 9

# MoltsPay: Universal Payment Protocol for AI Agents

**Version 2.0 | March 2026**


## Abstract

MoltsPay introduces the **Universal Payment Protocol (UPP)** - a unified payment abstraction layer that enables AI agents to pay for services across multiple blockchains using a single API. Unlike existing solutions that are locked to specific chains or protocols, UPP seamlessly integrates four distinct payment protocols (x402, MPP, Solana Pay-for-Success, and BNB Pre-Approval) behind one interface, supporting 7 blockchain networks across two virtual machines (EVM and SVM).

This whitepaper describes the core innovation of MoltsPay: how the Universal Payment Protocol abstracts away chain-specific complexity, enabling true multi-chain agent commerce without requiring clients or servers to implement multiple payment flows.


---

## 1. Introduction

### 1.1 The Problem

AI agents are becoming autonomous economic actors, but the payment infrastructure is fragmented:

**Chain Fragmentation:** - Different blockchains have different payment mechanisms - x402 works on EVM chains with EIP-3009 support - Solana requires SPL token transfers with fee payers - Some chains (Tempo) have native gas-free models - Developers must implement separate flows for each chain

**Protocol Fragmentation:** - x402 requires external facilitators (Coinbase CDP) - MPP (Machine Payment Protocol) uses direct on-chain transfers - Some chains require pre-approvals, others use signatures - No unified way to handle all these protocols

**The Result:** Service providers must choose ONE chain, and clients must match. This limits the market for both sides.

### 1.2 Our Contribution: Universal Payment Protocol

MoltsPay solves this with the **Universal Payment Protocol (UPP)**:

**Key Innovation:** One API call works on any supported chain. The protocol layer handles all chain-specific details automatically.

```
# Same command, different chains, different underlying protocols
moltspay pay https://service.com api --chain base      # Uses x402
moltspay pay https://service.com api --chain solana    # Uses Solana SPL
moltspay pay https://service.com api --chain tempo_moderato  # Uses MPP
moltspay pay https://service.com api --chain bnb       # Uses BNB pre-approval
```

---

## 2. Universal Payment Protocol Architecture

### 2.1 Protocol Abstraction Layer

The UPP consists of four layers:

1. **Application Layer:** Client SDK and Server SDK with language bindings (Node.js, Python)
2. **Protocol Layer:** Universal Payment Protocol handling chain detection, protocol selection, signature generation, and payment verification

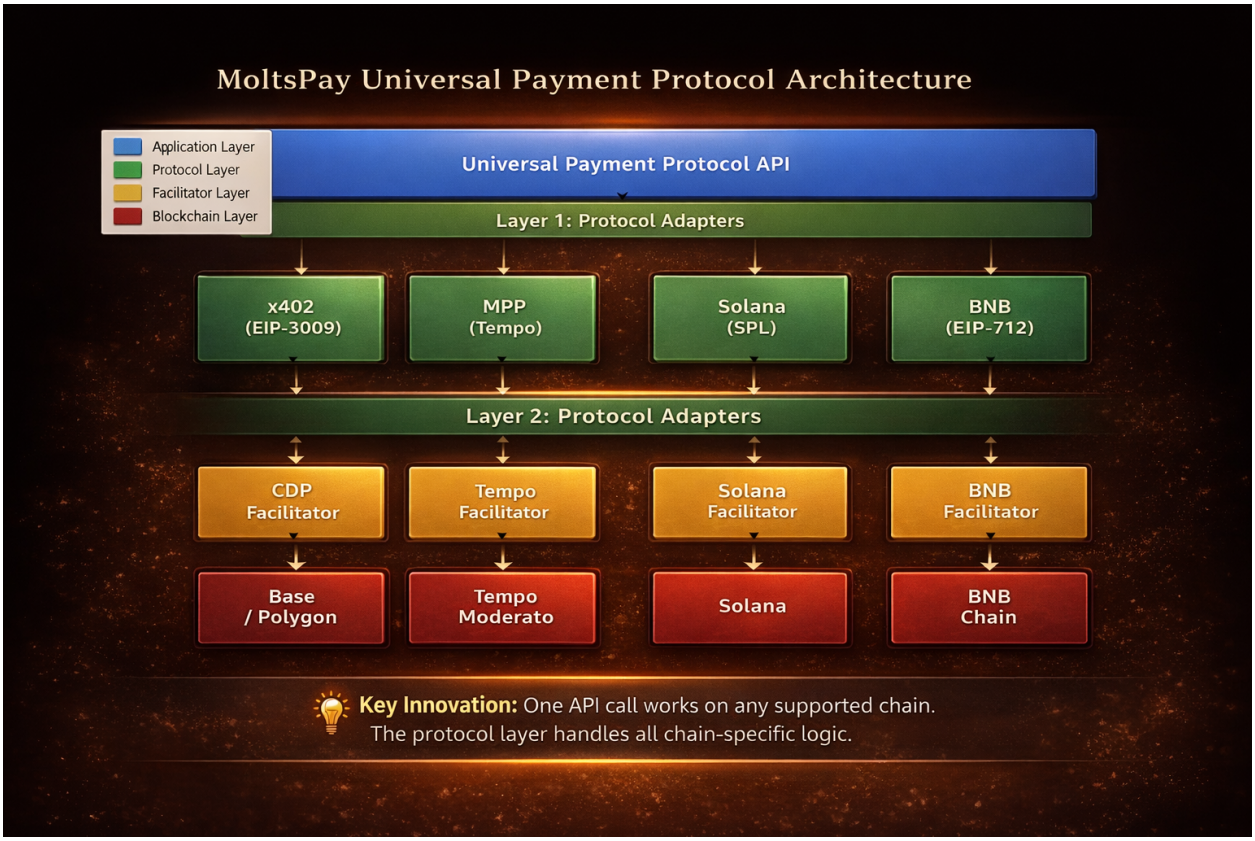


Figure 1: MoltsPay Architecture

3. **Facilitator Layer:** Chain-specific implementations (CDPFacilitator, TempoFacilitator, SolanaFacilitator, BNBFacilitator)
4. **Blockchain Layer:** On-chain settlement across EVM chains (Base, Polygon, BNB, Tempo) and SVM chains (Solana)

### 2.2 Protocol Selection Logic

When a payment is initiated, UPP automatically selects the appropriate protocol based on the target chain:

| Chain | Facilitator | Protocol |
|---|---|---|
| Base, Polygon, Base Sepolia | CDPFacilitator | x402 with EIP-3009 |
| Tempo Moderato | TempoFacilitator | MPP direct transfer |
| Solana, Solana Devnet | SolanaFacilitator | SPL with fee payer |
| BNB, BNB Testnet | BNBFacilitator | EIP-712 + pre-approval |

### 2.3 Unified Payment Flow

Despite different underlying protocols, the user-facing flow is identical:

1. **Client** sends `POST /execute` to Server
2. **Server** returns `402 Payment Required` with payment requirements
3. **Client** (via UPP) generates appropriate signature based on chain/protocol
4. **Client** retries with `X-Payment` header containing signature
5. **Server** (via UPP) selects correct facilitator, verifies signature, settles on-chain
6. **Server** returns `200 OK` with service result

---

## 3. Supported Protocols

### 3.1 x402 Protocol (Base, Polygon)

The x402 protocol uses HTTP 402 status codes with EIP-3009 signatures:

**How it works:** 1. Server returns `402 Payment Required` with payment details 2. Client signs EIP-3009 `transferWithAuthorization` 3. Client retries with `X-Payment` header containing signature 4. External facilitator (Coinbase CDP) executes on-chain transfer

**Characteristics:** - Gasless for both client and server - Requires external facilitator (CDP) - Supported tokens: USDC, USDT

### 3.2 MPP - Machine Payment Protocol (Tempo)

MPP is designed for gas-free chains where clients can pay directly:

**How it works:** 1. Server returns `402 Payment Required` with payment details 2. Client transfers TIP-20 tokens directly on-chain 3. Client includes transaction hash in `X-Payment` header 4. Server verifies on-chain receipt

**Characteristics:** - No gas fees (Tempo is natively gas-free) - No facilitator needed - Client executes transaction directly - Supported tokens: pathUSD

### 3.3 Solana Pay-for-Success Protocol

Solana uses a server-as-fee-payer model with pay-for-success semantics:

**How it works:** 1. Server returns `402 Payment Required` with Solana payment details 2. Client signs SPL token transfer authorization 3. Server executes service first 4. If service succeeds: Server submits transaction (pays fee) 5. If service fails: No transaction submitted (client keeps funds)

**Characteristics:** - Gasless for client (server pays fee) - Pay-for-success protects client - Separate Ed25519 wallet (not EVM-compatible) - Supported tokens: USDC

### 3.4 BNB Pre-Approval Protocol

BNB Chain doesn't support EIP-3009, so we use a pre-approval model:

**How it works:** 1. One-time: Client approves server's spender address 2. Server returns `402 Payment Required` with BNB payment details 3. Client signs EIP-712 payment intent 4. Server executes service first 5. If service succeeds: Server calls `transferFrom` (pays gas) 6. If service fails: No transfer (client keeps funds)

**Characteristics:** - One-time approval tx (then gasless) - Pay-for-success protects client - Server pays gas - Supported tokens: USDC, USDT

---

## 4. Multi-Chain Support

### 4.1 Supported Networks

| Chain | Chain ID | VM | Protocol | Status |
|---|---|---|---|---|
| Base | 8453 | EVM | x402 | Production |
| Polygon | 137 | EVM | x402 | Production |
| Solana | mainnet | SVM | Solana | Production |
| BNB Chain | 56 | EVM | BNB | Production |
| Tempo Moderato | 42431 | EVM | MPP | Testnet |
| Base Sepolia | 84532 | EVM | x402 | Testnet |
| Solana Devnet | devnet | SVM | Solana | Testnet |
| BNB Testnet | 97 | EVM | BNB | Testnet |

### 4.2 Cross-VM Unification

MoltsPay is unique in supporting both EVM and SVM under one API:

**EVM Chains:** - secp256k1 cryptography - ERC-20 / TIP-20 token standard - Single wallet address across all EVM chains

**Solana (SVM):** - Ed25519 cryptography - SPL token standard - Separate wallet (different key type)

**Unified Wallet Management:**

```
$ moltspay status

EVM Wallet: 0x1234...abcd
  Base:     2.50 USDC
  Polygon:  1.00 USDC
  BNB:      5.00 USDC

Solana Wallet: 7xKp...9mNq
  Solana:   3.00 USDC
```

### 4.3 Server Multi-Chain Configuration

Servers can accept payments on multiple chains with one config:

```
{
  "provider": {
    "name": "My AI Service",
    "wallet": "0x...",
    "solana_wallet": "7xKp...",
    "chains": [
      {"chain": "base", "tokens": ["USDC"]},
      {"chain": "polygon", "tokens": ["USDC"]},
      {"chain": "solana", "tokens": ["USDC"]},
      {"chain": "bnb", "tokens": ["USDC", "USDT"]}
    ]
  }
}
```

The server automatically handles payments from any configured chain using the appropriate protocol.

---

## 5. Facilitator Architecture

### 5.1 Facilitator Registry

The Facilitator Registry routes payments to the correct handler:

```
class FacilitatorRegistry {
  private facilitators: Map<string, Facilitator>;

  getFacilitator(network: string): Facilitator {
    if (network in ['base', 'polygon', 'base_sepolia']) {
      return this.facilitators.get('cdp');
    }
    if (network in ['solana', 'solana_devnet']) {
      return this.facilitators.get('solana');
    }
    if (network in ['bnb', 'bnb_testnet']) {
      return this.facilitators.get('bnb');
    }
    if (network === 'tempo_moderato') {
      return this.facilitators.get('tempo');
    }
  }
}
```

### 5.2 External vs Self-Hosted Facilitators

| Facilitator | Type | Chains | Gas Model |
|---|---|---|---|
| CDP | External | Base, Polygon | CDP pays |
| Solana | Self-hosted | Solana | Server pays |
| BNB | Self-hosted | BNB | Server pays |
| Tempo | Self-hosted | Tempo | Gas-free |

**Self-hosted facilitators enable:** - True decentralization (no third-party dependency) - Support for any chain (just implement the interface) - Cost control (only pay actual gas) - Censorship resistance

### 5.3 Facilitator Interface

All facilitators implement the same interface:

```
interface Facilitator {
  name: string;
  supportedNetworks: string[];

  verify(payload: PaymentPayload): Promise<VerifyResult>;
  settle(payload: PaymentPayload): Promise<SettleResult>;
  supportsNetwork(network: string): boolean;
}
```

This allows new chains to be added by implementing a single interface.

---

## 6. Security Model

### 6.1 Client Security

- **Spending Limits:** Per-transaction and daily limits prevent runaway spending
- **Wallet Isolation:** EVM and Solana wallets are separate files
- **Local Keys:** Private keys never leave the client device
- **Signature-Only:** Client only signs; never executes transactions directly (except MPP)

### 6.2 Server Security

- **On-chain Verification:** All payments verified on-chain before service execution
- **Pay-for-Success:** Solana/BNB only settle if service succeeds
- **No Custody:** Server never holds user funds
- **Multi-sig Support:** Provider wallets can be multi-sig

### 6.3 Protocol Security

| Protocol | Client Risk | Server Risk |
|---|---|---|
| x402 | Signature can only transfer specified amount | CDP verifies before settlement |
| MPP | Client pays first (higher risk) | Server verifies on-chain |
| Solana | Pay-for-success protects client | Server pays gas on success |
| BNB | Pre-approval limited to server's spender | Server pays gas on success |

---

## 7. Implementation

### 7.1 SDK Support

**Node.js:**

```
npm install moltspay
```

**Python:**

```
pip install moltspay
```

### 7.2 Client Usage

```
import { MoltsPay } from 'moltspay';

const client = new MoltsPay();

// Pay on any chain - UPP handles the rest
const result = await client.pay({
  url: 'https://service.com',
  service: 'text-to-video',
  chain: 'base',  // or 'solana', 'bnb', 'tempo_moderato'
  params: { prompt: 'A cat dancing' }
});
```

### 7.3 Server Usage

```
import { MoltsPayServer } from 'moltspay/server';

const server = new MoltsPayServer('./my-skill');
server.listen(8402);

// Server automatically accepts payments on all configured chains
// UPP handles protocol selection and verification
```

---

## 8. Future Work

### 8.1 Additional Protocols

- **Arbitrum/Optimism:** Extend x402 support
- **Sui/Aptos:** New VM support (Move-based)

### 8.2 Protocol Improvements

- **Cross-chain Payments:** Pay on Chain A, receive on Chain B
- **Streaming Payments:** Real-time micro-payments

### 8.3 Decentralization

- **On-chain Facilitator Registry:** Decentralized facilitator discovery
- **Facilitator Staking:** Economic security for facilitators

---

## 9. Conclusion

The Universal Payment Protocol represents a fundamental shift in how AI agents interact with paid services. By abstracting four distinct payment protocols behind a single API, MoltsPay eliminates the fragmentation that has limited multi-chain commerce.

**Key Contributions:**

1. **Protocol Unification:** x402, MPP, Solana, and BNB protocols unified under one interface

2. **Cross-VM Support:** First solution to support both EVM and SVM chains
3. **Self-Hosted Facilitators:** True decentralization without third-party dependencies
4. **Gasless Experience:** Users never pay gas on any supported chain
5. **Pay-for-Success:** Protection for clients on self-hosted facilitator chains

As AI agents become more autonomous, the need for universal payment infrastructure will only grow. MoltsPay provides that infrastructure today, enabling any agent to pay any service on any supported chain with a single API call.

---

## References


- x402 Protocol Specification
- EIP-3009: Transfer With Authorization
- EIP-712: Typed Structured Data Hashing
- Solana SPL Token Program
- Tempo TIP-20 Standard
- Coinbase Developer Platform


---



**Repository:** https://github.com/Yaqing2023/moltspay

**Website:** https://moltspay.com