x402 Payment Protocol: Technical Documentation

using pandoc Convert to PDF with nice styling

pandoc x402-technical-documentation.md
-o x402-technical-documentation.pdf
–pdf-engine=xelatex
-V geometry:margin=1in
-V colorlinks=true
-V linkcolor=blue

How Penny.io Enables Instant Micropayments on Base

MAKE SURE TO ADD SOLANA SUPPORT

Table of Contents

What is x402?

x402 is an HTTP payment protocol that enables instant micropayments on Layer 2 networks.

Inspired by HTTP's 402 Payment Required status code (reserved but never implemented), x402 brings native payment functionality to the web. Instead of requiring readers to connect payment methods, create accounts, or manage subscriptions, x402 allows wallet-to-wallet micropayments with a single signature.

Think of it as HTTP, but for money.

Key Features

- Instant Verification - Payments verified in ~2 seconds (vs 15+ seconds for on-chain transactions)
- Micropayment Optimized - Designed for $0.01-$1.00 transactions
- No Platform Custody - Direct wallet-to-wallet transfers
- Signature-Based - Uses EIP-712 typed signatures for authorization
- Layer 2 Native - Built for Base, Optimism, Arbitrum, and other L2s
- Open Standard - Fully open-source protocol

Why x402 Matters

Traditional blockchain payments have a critical flaw: every transaction requires on-chain submission, which means:

- ⊠ High gas fees ($0.10-$0.50 per transaction, even on L2s)
- ⊠ Slow confirmation times (15-60 seconds per payment)
- ⊠ Poor UX (multiple wallet popups, long waits)

x402 solves this with authorization-based payments:

[OK] User signs an authorization (like signing a check)

[OK] Platform verifies the signature instantly (off-chain)

[OK] Content unlocks immediately

[OK] Settlement happens later, in batches (optional)

This makes micropayments economically viable for the first time.

How It Works

The x402 Flow (High-Level)

```
 Reader              Frontend                Backend

(Wallet)




    1. Click Purchase

              >



                          2. POST /purchase

                            (no X-PAYMENT)

                                     >



                          3. 402 Requirements

                         <



    4. Sign Authorization

      (ONE popup!)

   <
```

```
     5. Signature

               >


                                  6. POST /purchase

                                    + X-PAYMENT header

                                               >


                                  7. Verify Signature


                                  8. [OK] Valid!


                                  9. Grant Access

                                 <


     10. Content Unlocked

    <
```

Three-Step Purchase Flow

```
 Step 1: Verify Authorization

 [OK] Valid signature? -> Continue

 [X] Invalid? -> Return 400 error


                        ↓


 Step 2: Settle Payment (Optional)

 [OK] CDP transfers USDC (gasless)
```

```
[OK] Or: Record authorization for batch settlement

                    ↓

Step 3: Grant Access

[OK] Record payment in database

[OK] Unlock content for user

[OK] Return success response
```

Payment Flow Diagram

Complete x402 Payment Flow (Penny.io Implementation)

```
 Reader              Frontend                Backend              Blockchain

(Wallet)                                                          (Base L2)




     1. Click Purchase

              >



                              2. POST /purchase

                                (no X-PAYMENT)

                                        >



                              3. 402 Requirements

                             <
 4. Sign Authorization
   (ONE popup!)
<


 5. Signature
           >


                       6. POST /purchase
                         + X-PAYMENT header
                                  >



                                               7. Verify with
                                                  CDP facilitator


                                               8. [OK] Valid!


                                               9. Submit authorization
                                                  on-chain (platform
                                                  wallet pays gas)
                                                          >


                                               10. Transaction hash
                                              <


                                               11. Update DB with
                                                   tx hash
```

```
                              12. Success + receipt

                             <



       13. Content unlocked

      <
```

Implementation Details

Key Components

Component	File	Role
Payment Initiator	frontend/src/pages/Article.tsx	User clicks “Purchase” -> calls x402PaymentService
Payment Orchestrator	frontend/src/services/x402PaymentService.ts	1) GET requirements 2) Sign payment 3) Send X-PAYMENT header
Payment Processor	backend/src/routes.ts	1) Return 402 requirements 2) Verify signature 3) Record payment
Payment Tracker	backend/src/database.ts	Stores payment in PostgreSQL (prevents duplicate payments)
Facilitator Selector	backend/src/routes.ts	Chooses CDP vs public facilitator based on API keys

---

Authorization vs Settlement

The Critical Distinction

When a user purchases an article on Penny.io, here’s what actually happens:

What You Might Think Happened:

1. [OK] User signed transaction
2. [OK] $0.12 USDC transferred on-chain
3. [OK] Author received USDC in their wallet
4. [OK] Transaction hash recorded

What Actually Happened:

1. [OK] User signed an authorization (like signing a check)
2. [OK] Backend verified authorization is valid
3. [OK] Backend recorded “this user CAN pay $0.12 USDC”
4. NO on-chain transaction happened (yet!)
5. USDC is STILL in user’s wallet

Analogy: Checks vs Cash

Traditional Blockchain	x402 Authorization

Handing someone cash (instant transfer) | Writing a signed check (promise to pay) |

Money moves immediately | Money moves when check is cashed |

$0.50 gas per payment | $0.0001 gas when batching many checks |

What's Recorded in the Database

```
{

  articleId: 123,

  userAddress: "0x742d35...",

  amount: 0.12,

  paymentVerified: true,

  transactionHash: null  // Will be populated after settlement

}
```

[OK] User signed a valid authorization

[OK] User HAS the USDC (facilitator verified)

[OK] Authorization CAN be settled on-chain anytime

[X] Money hasn't actually moved yet

When Does Settlement Happen?

Option 1: Immediate Settlement (Current Implementation)

- Backend submits authorization on-chain immediately
- Platform wallet pays gas (~$0.0001 on Base)
- USDC transfers to author's wallet
- Transaction hash recorded

Option 2: Batch Settlement (Future Optimization)

- Collect 100-1,000 authorizations
- Submit all in a single transaction
- Gas cost divided across all payments
- Even lower per-payment cost

Gas Costs & Scalability

Who Pays Gas?

Anyone can submit the authorization on-chain!

Think of it like a signed check:

- The check is signed by the buyer ([OK] valid signature)
- But ANYONE can walk into the bank and deposit it

- Whoever deposits it = whoever pays the gas

In Penny.io's implementation, the platform wallet pays gas to provide a seamless user experience.

Gas Cost Reality Check

Scenario	Gas Price	Cost per Payment	1,000 Payments	10,000 Payments
Base L2 (typical)	~0.001 Gwei	$0.00005	$0.05	$0.50
Base L2 (congested)	~0.01 Gwei	$0.0001	$0.10	$1.00

At scale:

- 10,000 purchases = $0.50-$1.00 total gas
- If avg article = $0.10, you earned $1,000
- Gas = 0.05-0.1% of revenue

This is why x402 makes micropayments viable.

Code Examples

1. Building Payment Requirements (Backend)

```
// backend/src/routes.ts:152-194

function buildPaymentRequirement(

  article: Article,

  req: Request,

  network: SupportedX402Network

): PaymentRequirements {

  const priceInCents = Math.round(article.price * 100);

  const priceInMicroUSDC = (priceInCents * 10000).toString();

  const asset = network === 'base'

    ? '0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913'  // USDC on Base

    : '0x036CbD53842c5426634e7929541eC2318f3dCF7e'; // USDC on Base Sepolia

  const resourceUrl = `${req.protocol}://${req.get('host')}/api/articles/${article.id}/purchase?network

  return {

    scheme: 'exact',

    network,
    maxAmountRequired: priceInMicroUSDC,
    resource: resourceUrl,
    description: `Purchase access to: ${article.title}`,
    mimeType: 'application/json',
    payTo: article.authorAddress,  // ← Dynamic recipient!
    maxTimeoutSeconds: 60,
    asset,
    outputSchema: {
      input: { type: 'http', method: 'POST', discoverable: true }
    },
    extra: {
      name: 'USDC',
      version: '2',
      title: `Purchase: ${article.title}`,
      category: article.categories?.[0] || 'content',
      serviceName: 'Penny.io Article Access',
      pricing: {
        currency: 'USD',
        amount: article.price.toString(),
        display: `$${article.price.toFixed(2)}`
      }
    }
  };
}
```

Key Features:

- Dynamic Pricing - Each article can have a different price
- Dynamic Recipients - Payments go directly to article authors
- Network Flexibility - Supports both mainnet (Base) and testnet (Base Sepolia)

---

2. Payment Verification (Backend)

```
// backend/src/routes.ts:519-574

router.post('/articles/:id/purchase', criticalLimiter, async (req, res) => {

  const articleId = parseInt(req.params.id);

  const article = await db.getArticleById(articleId);

  if (!article) {

    return res.status(404).json({ success: false, error: 'Article not found' });

  }

  const paymentRequirement = buildPaymentRequirement(article, req, networkPreference);

  const paymentHeader = req.headers['x-payment'];

  // Step 1: If no X-PAYMENT header, return 402 with requirements

  if (!paymentHeader) {

    return res.status(402).json({

      x402Version: 1,

      error: 'X-PAYMENT header is required',

      accepts: [paymentRequirement]

    });

  }

  // Step 2: Decode and parse the payment payload

  let paymentPayload: PaymentPayload;

  try {

    const decoded = Buffer.from(paymentHeader as string, 'base64').toString('utf8');

    paymentPayload = PaymentPayloadSchema.parse(JSON.parse(decoded));

  } catch (error) {

    return res.status(400).json({ success: false, error: 'Invalid x402 payment header' });

  }
  // Step 3: Verify amounts match
  const requiredAmount = BigInt(paymentRequirement.maxAmountRequired);
  const providedAmount = BigInt(paymentPayload.payload.authorization.value);
  if (providedAmount < requiredAmount) {
    return res.status(400).json({ success: false, error: 'Insufficient payment amount' });
  }
  // Step 4: Verify signature with facilitator (CDP or public)
  const verification = await verifyWithFacilitator(paymentPayload, paymentRequirement);
  if (!verification.isValid) {
    return res.status(400).json({
      success: false,
      error: `Payment verification failed: ${verification.invalidReason}`
    });
  }
  // Step 5: Verify recipient matches article author
  const paymentRecipient = normalizeAddress(paymentPayload.payload.authorization.to);
  if (paymentRecipient !== article.authorAddress) {
    return res.status(400).json({ success: false, error: 'Payment recipient mismatch' });
  }
  // Step 6: Record payment and grant access
  // ... (continues with settlement and database recording)
});
```

Security Checks:

1. [OK] Amount Verification - Ensures payment matches article price
2. [OK] Signature Verification - Uses CDP facilitator for cryptographic validation
3. [OK] Recipient Verification - Ensures payment goes to correct author
4. [OK] Duplicate Prevention - Database UNIQUE constraint prevents double-spending

---

3. Frontend Payment Orchestration

```
// frontend/src/services/x402PaymentService.ts:161-199
async purchaseArticle(
  articleId: number,
  walletClient: WalletClient,
  networkOverride?: SupportedNetwork
): Promise<PaymentResponse> {
  try {
    // Step 1: Initial request without payment (triggers 402)
    const initialResponse = await this.attemptPayment(
      `/articles/${articleId}/purchase`,
      undefined,
      networkOverride
    );
    if (initialResponse.paymentRequired && initialResponse.paymentRequired.accept) {
      // Step 2: Create signed payment header
      const encodedHeader = await this.createPaymentHeaderFromRequirements(
        initialResponse.paymentRequired,
        walletClient
      );
      // Step 3: Retry request with X-PAYMENT header
      const response = await fetch(
        this.buildRequestUrl(`/articles/${articleId}/purchase`, networkOverride),
        {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'X-PAYMENT': encodedHeader
          }
        }
      );
      const result = await response.json();
      if (response.ok && result.success) {
        return {
          success: true,
          receipt: result.data?.receipt || 'Payment processed',
          encodedHeader,
          rawResponse: result
        };
      }
      return {
        success: false,
        error: result?.error || `Payment failed with status ${response.status}`,
        encodedHeader
      };
    }
    throw new Error('No payment requirements returned from server');
  } catch (error) {
    console.error('Purchase failed:', error);
    return {
      success: false,
      error: error instanceof Error ? error.message : 'Unknown error occurred'
    };
  }
}
```

User Experience:

- Single Signature - User only signs once (no multiple popups)
- Fast Verification - ~2 second round-trip time
- Error Handling - Clear error messages for debugging

---

4. Payment Tracking (Database)

```
// backend/src/database.ts:635-659

async recordPayment(

  articleId: number,

  userAddress: string,

  amount: number,

  transactionHash?: string

): Promise<boolean> {

  const normalizedUserAddress = tryNormalizeAddress(userAddress) ?? userAddress;

  const { error } = await supabase.from('payments').insert({

    article_id: articleId,

    user_address: normalizedUserAddress,

    amount,

    transaction_hash: transactionHash,

    payment_verified: true,

    created_at: new Date().toISOString(),

  });

  if (error) {

    // Ignore duplicate payment errors (UNIQUE constraint)

    if (error.code === '23505') {

      console.log(`Payment already recorded for article ${articleId} by ${normalizedUserAddress}`);

      return false;

    }
    throw error;

  }

  return true;

}
```

Database Schema:

```
CREATE TABLE payments (

  id SERIAL PRIMARY KEY,

  article_id INTEGER NOT NULL REFERENCES articles(id),

  user_address TEXT NOT NULL,

  amount DECIMAL(10, 2) NOT NULL,

  transaction_hash TEXT,

  payment_verified BOOLEAN DEFAULT FALSE,

  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

  UNIQUE(article_id, user_address)  -- Prevents duplicate payments

);
```

Key Features:

- Persistent Storage - Payments tracked across server restarts
- Duplicate Prevention - UNIQUE constraint on (article_id, user_address)
- Transaction Tracking - Records on-chain settlement hash when available

---

Security Considerations

1. Signature Security

Question: Can users pay without the wallet popup?

Answer: No. This is a fundamental blockchain security principle.

In blockchain, a signature = spending money. You cannot spend someone's money without their explicit approval. The wallet popup MUST exist for security.

However, x402 IS optimized:

- Regular transaction: Sign -> Wait for blockchain confirmation -> Slow (15+ seconds)
- x402 signature: Sign -> Instant verification -> Fast (~2 seconds)

The popup is quick, but it exists for your protection.

2. Facilitator Trust Model

Penny.io uses Coinbase Developer Platform (CDP) as the payment facilitator:

- [OK] Signature Verification - Validates EIP-712 signatures cryptographically
- [OK] Balance Checking - Ensures user has sufficient USDC before authorization
- [OK] Gasless Settlement - CDP submits transactions on-chain (optional)

Fallback Strategy:

- If CDP keys configured -> Uses CDP facilitator
- If no CDP keys -> Uses public facilitator
- If facilitator fails -> Payment rejected (safe failure mode)

3. Duplicate Payment Prevention

Database-level protection:

```
UNIQUE(article_id, user_address)
```

This ensures:

☒ User cannot pay twice for the same article

☒ Backend cannot accidentally record duplicate payments

- [OK] First payment wins (idempotent operation)

4. Rate Limiting

```
const criticalLimiter = rateLimit({

  windowMs: 15 * 60 * 1000,  // 15 minutes

  max: 10,                    // 10 requests max

  message: { success: false, error: 'Too many payment attempts' }

});
```

Prevents:

☒ Payment spam attacks

☒ Brute-force signature guessing

☒ API abuse

---

402 Payment Required Response

Example Payload

```
{

  "x402Version": 1,

  "error": "X-PAYMENT header is required",

  "accepts": [
    {
      "scheme": "exact",
      "network": "base-sepolia",
      "maxAmountRequired": "120000",
      "resource": "http://localhost:3001/api/articles/2/purchase",
      "description": "Purchase access to: Building Scalable Web3 Applications",
      "mimeType": "application/json",
      "payTo": "0x25A6bd85966E9dE449cf36F7f6A033944C8bB5D8",
      "maxTimeoutSeconds": 60,
      "asset": "0x036CbD53842c5426634e7929541eC2318f3dCF7e",
      "outputSchema": {
        "input": {
          "type": "http",
          "method": "POST",
          "discoverable": true
        }
      },
      "extra": {
        "name": "USDC",
        "version": "2",
        "title": "Purchase: Building Scalable Web3 Applications",
        "category": "Technology",
        "serviceName": "Penny.io Article Access",
        "serviceDescription": "Unlock full access to the article",
        "pricing": {
          "currency": "USD",
          "amount": "0.12",
          "display": "$0.12"
        }
      }
    }
  ]
}
```

Field Explanations:

Field	Description
scheme	Payment scheme (exact = fixed price)
network	Target blockchain (base, base-sepolia, etc.)
maxAmountRequired	Price in micro-USDC (120000 = $0.12)
resource	API endpoint to retry with payment
payTo	Author’s wallet address (dynamic per article)
asset	USDC contract address on target network
extra.pricing	Human-readable price display

CDP Integration

Using Coinbase Developer Platform

Penny.io integrates with CDP for:

1. Signature Verification - Validates EIP-712 authorization signatures
2. Balance Checking - Ensures user has sufficient USDC
3. Gasless Settlement - CDP submits transactions on-chain (optional)

Configuration:

```
const facilitator = process.env.CDP_API_KEY_NAME && process.env.CDP_API_KEY_PRIVATE_KEY
  ? 'cdp'  // Use CDP if keys are configured
  : 'public';  // Otherwise use public facilitator
```

Benefits:

- [OK] “Powered by Coinbase” trust signal
- [OK] Enterprise-grade security and uptime
- [OK] Gasless transactions for users (platform pays gas)

Conclusion

x402 makes micropayments economically viable by separating authorization from settlement.

- Users sign once and get instant access
- Authors receive direct wallet-to-wallet payments
- Platform pays minimal gas costs ($0.0001 per payment on Base)

This is the future of content monetization.

Learn More

- Try Penny.io: https://penny.io
- GitHub Repository: https://github.com/mbeliogl/Penny.io
- x402 Specification: https://x402.org
- Coinbase Developer Platform: https://cdp.coinbase.com

Built on Base • Powered by x402 • Open Source

Last Updated: November 2025