



OST

Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

x402

Thomas Bocek

01.10.2026

Learning Goals

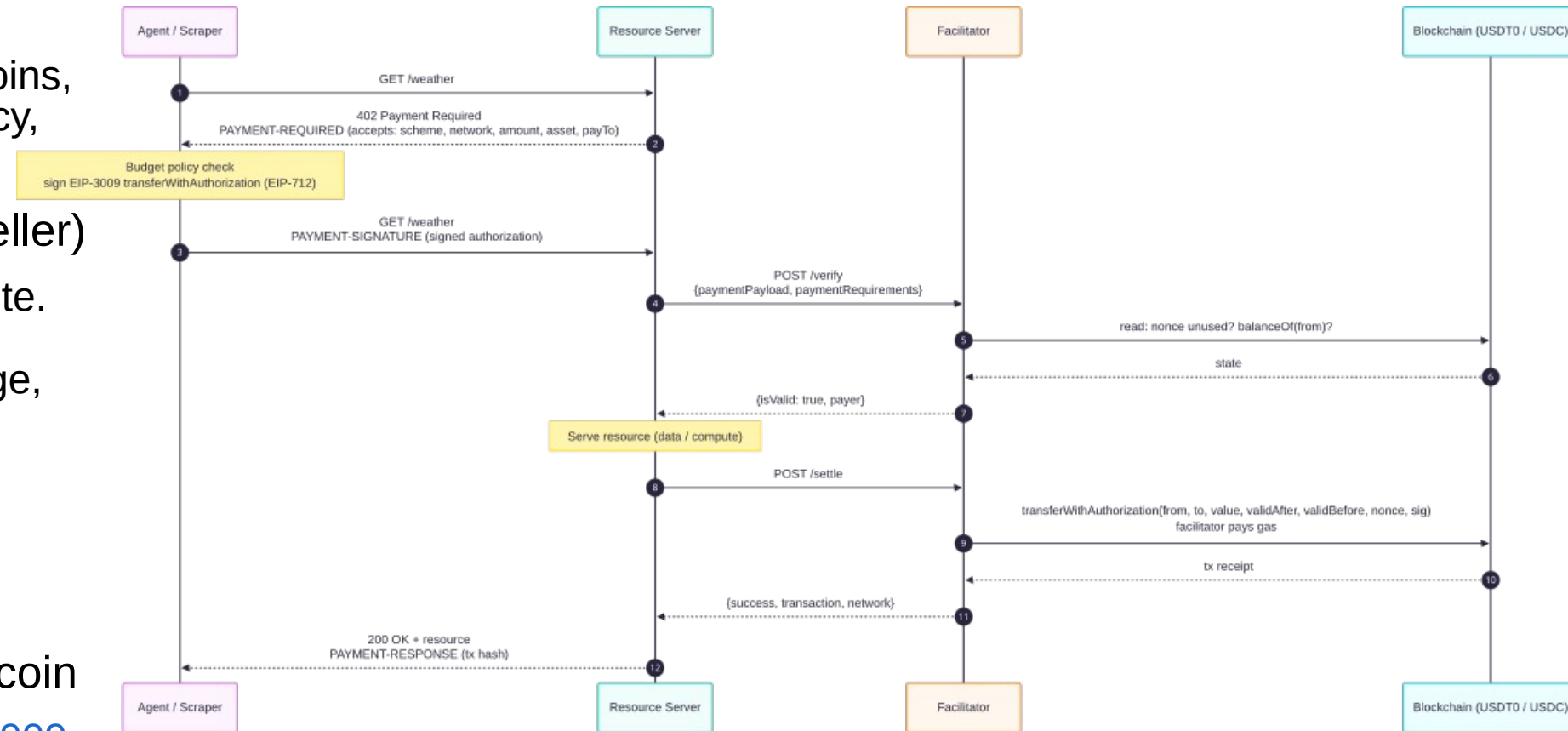
- Lecture 4 – x402
 - HTTP 402 and the x402 protocol: roles, headers, schemes
 - Free tier vs. paid tier: rate limiting and why 402 must own the decision
 - Gasless stablecoin payments: EIP-3009 transferWithAuthorization
 - Facilitator internals: /supported, /verify, /settle
 - Agent client: budget policy, pay, retry
 - Where x402 sits next to UCP (agentic commerce)
- Challenge Task HS 2026
 - Resource server: free tier, then 402 with price, network, asset, recipient
 - Agent/scrapper: reads 402, checks budget, signs, retries
 - Facilitator: verify + settle (groups: own implementation)
 - Stablecoin on a public testnet, EIP-3009 or equivalent
 - Simple frontend for each of the three components

Why 402? APIs for machines, not people

- Problem
 - API: sign-up, API key, subscription
 - Agents, scrapers: no e-mail, no card, no KYC
 - May have a wallet
 - Cards: minimum amounts, ~2–3% + fixed fee, chargebacks → no 0.001 \$ requests
 - Credit Cards: promise to pay
 - Settled stablecoin transfer is proof of payment
 - Servers refuse (401/403/429)
 - What if they could ask for money instead
- HTTP 402 Payment Required
 - HTTP/1.1 since [1997](#), "reserved for future use"
 - Semantics never standardized
 - Earlier attempts: micropayment schemes, [L402](#) (Lightning), Web Monetization
 - [x402](#): Coinbase, whitepaper May 2025; now the x402 Foundation (Coinbase + Cloudflare, Linux Foundation), v2 spec
 - Coinbase numbers [[link](#)], April 2026: ~69k agents, ~165M transactions, ~50M \$ cumulative, ~0.30 \$ per transaction

Stakeholders and the x402 flow

- Agent (buyer)
 - Wallet with stablecoins, applies budget policy, signs, retries
- Resource server (seller)
 - Serves data/compute. Rate limits free tier, issues 402 challenge, verifies and settles
- Facilitator
 - Verifies signatures, submits, pays gas
- Blockchain + stablecoin
 - Must support [EIP-3009](#)



API Server – pay for weather

// x402 v2 challenge. Fill in your chain, token contract and payTo.

```
const challenge = `{
  "x402Version":2,
  "resource":{"url":"http://localhost:8080/weather"},
  "accepts":[{"scheme":"exact","network":"eip155:84532",
  "amount":"1000","asset":"0x036CbD53842c5426634e7929541eC2318f3dCF7e",
  "payTo":"0x0000000000000000000000000000000000000000000000000000000000000001",
  "maxTimeoutSeconds":60,"extra":{"name":"USDC","version":"2"}}}]`
```

```
func handler(w http.ResponseWriter, r *http.Request) {
    // 1. free tier: if freeTier.Allow(clientKey(r)) { serve(w, r); return }
    // 2. paid retry: if r.Header.Get("PAYMENT-SIGNATURE") != "" {
    //             facilitator.Verify(r) -> POST /verify
    //             serve(w, r)          -> real data
    //             facilitator.Settle(r) -> POST /settle, PAYMENT-RESPONSE
    //             return }
    // 3. otherwise: challenge
    w.Header().Set("PAYMENT-REQUIRED",
        base64.StdEncoding.EncodeToString([]byte(challenge)))
    w.WriteHeader(http.StatusPaymentRequired) // 402
}
```

Free tier: rate limit at the proxy, 402 in the app

- Naive approach: rate limit in caddy (DSy)
 - Anyone with the header: "PAYMENT-SIGNATURE x" can get free stuff
 - Rate limiter only works if it's coupled to verification
- Better approach: rate limiter in the server
 - Coupled with verification

```
api.example.com {
  rate_limit {
    zone free {
      match { not header PAYMENT-SIGNATURE * }
      key    {client_ip}
      events 20
      window 1h
    }
  }
  handle_errors 429 {          # turn "wait" into "pay"
    header PAYMENT-REQUIRED {env.X402_CHALLENGE}
    respond 402
  }
  reverse_proxy app:8080
}
```

```
var free = map[string]int{} // ip -> used, reset 1h

func handler(w http.ResponseWriter, r *http.Request) {
  sig := r.Header.Get("PAYMENT-SIGNATURE")
  if sig == "" {                                // unpaid path
    if free[ip(r)] >= 20 { reply402(w); return }
    free[ip(r)]++                               // count free use
    serve(w, r); return
  }
  payer, ok := facilitator.Verify(sig) // proof
  if !ok { reply402(w); return }       // forged: 402
  serve(w, r)                          // paid: not counted
  facilitator.Settle(sig)
  // block pricing: credit[payer] += n on settle,
  // consume credit[payer] before the next 402
}
```

Three building blocks: ERC-20, EIP-712, EIP-3009

- ERC-20 (next lecture): A token is a contract with mapping(address => uint256) balanceOf and transfer(to, value), ...
- EIP-712: sign structured data
 - Signature overstruct, bound to chain/contract
 - Wallet shows readable fields, not a hex blob
 - chainId + verifyingContract prevent replay on another chain or token
 - Any contract can recover the signer from the digest
- EIP-3009: transfer by signature
 - Contract verifies signature and moves the balance
 - Payer signs offline, facilitator submits the TX and pays gas (no account abstraction yet)
 - Nonce is a random bytes32, single use → attack protection, also validAfter / validBefore
 - msg.sender is not checked

```
domain = { name: "USDC", version: "2",  
  chainId: 84532, verifyingContract: asset }  
struct = TransferWithAuthorization(address from,  
  address to, uint256 value, uint256 validAfter,  
  uint256 validBefore, bytes32 nonce)  
digest = keccak256(0x1901 || hash(domain)  
  || hash(struct, values))  
(v, r, s) = sign(agentKey, digest) // signTypedData
```

```
function transferWithAuthorization(address from,  
  address to, uint256 value, uint256 validAfter,  
  uint256 validBefore, bytes32 nonce,  
  uint8 v, bytes32 r, bytes32 s) external {  
  require(now > validAfter && now < validBefore);  
  require(!authorizationState[from][nonce]);  
  require(ecrecover(digest, v, r, s) == from);  
  authorizationState[from][nonce] = true;  
  _transfer(from, to, value); // balances mapping  
}
```

Settlement: EIP-3009 – and why "USDT" is not one thing

- Agent signs, facilitator calls `transferWithAuthorization(from,to,value,...)`
- Contract checks signature, window and that nonce is unused (`authorizationState`) → replay-safe on-chain
- `receiveWithAuthorization` when called from a contract (front-running)
- Same ticker, different contracts, different EIP-712 domains (USDT)
- Testnet reality: USDC on Sepolia supported by x402.org facilitators
 - You also can deploy your own test-USD token

```
// EIP-712 typed data, signed by the agent
TransferWithAuthorization(
  address from,          // agent wallet
  address to,            // payTo: 0x...0001
  uint256 value,         // amount: 1000 (0.001)
  uint256 validAfter,    // 0 or now
  uint256 validBefore,   // now + maxTimeoutSeconds
  bytes32 nonce)        // random, one-time
// domain: name/version = extra,
// chainId = network, verifyingContract = asset
```

Token	EIP-3009	Notes
USDC (Circle)	yes	x402 default; domain "USDC" v2 on most chains, "USD Coin" on some
USDT, Ethereum (2017)	no	Plain ERC-20, no permit either → approve + transferFrom, agent pays gas
USDT0 (Tether, 2025)	yes	LayerZero OFT, EIP-3009 + 2612; domain name "USD₦0" (U+20AE!)

Facilitator: three endpoints, one hot wallet

- GET /supported

- Lists (x402Version, scheme, network)
 - Server check this and advertise is supported

```
{"kinds":[{"x402Version":2,
"scheme":"exact",
"network":"eip155:84532"}]}
```

- POST /verify

- In: paymentPayload + paymentRequirements.
Pure read, no tx. Checks:
 - Scheme/network supported, x402Version matches
 - EIP-712 signature recovers to from (EOA; EIP-1271/6492 for smart wallets)
 - ...

→ {"isValid":true,"payer":"0x..."}

- POST /settle

- Re-verify, send transferWithAuthorization from own EOA
- Receipt in a block (Base ~2 s, no finality wait)
→ success + tx hash, server delivers
- Pays gas → funded hot wallet, nonce queue
- Broadcast but no receipt → settlement_pending + tx hash; facilitator polls the hash, server retries /settle, nobody re-broadcasts
- Same payload twice → same result (cache by nonce)

```
{"success":true,"transaction":"0x...",
"network":"eip155:84532"}
```

Security

- Payer trusts nobody
 - Signature bound to payTo, value, validBefore
 - Nobody can redirect or increase it
 - Worst case: one micropayment without service
- Server trusts the facilitator
 - Wrong "/valid" or "/settle" → server has served, no tokens arrived
 - Mitigation: check tx hash on-chain, or run your own
- Facilitator pays gas → others can waste its money
 - Payer signs, then moves funds → transferWithAuthorization fails on balance, tx reverts, facilitator still pays gas
 - /verify spam = RPC cost
 - Needs fees, or a strategic operator (Coinbase, token issuer)
 - Custodial / prepaid -
 - Non-custodial 2-of-2

Agent client: a signature is money, policy before signing

- Budget policy (CT requirement 6)
 - Allowlist networks and asset addresses – the server can put any token in accepts
 - Cap per request, per host, per day → over cap: do not sign
 - Amounts are atomic units: check decimals (USDC/USDT: 6)
 - Short validBefore – a signed authorization is money until it expires
 - Sign only for the resource that returned the 402, to the payTo it gave
- Per request vs. per block
 - Per request: exact scheme, one settle per call – simple, gas per call
 - Per block: batch-settlement scheme, or one larger payment + server-side credit – cheaper, server tracks balance

```
resp := get(url)
if resp.Status != 402 { return resp }
req := decode(resp.Header["PAYMENT-REQUIRED"])
opt := pick(req.Accepts, policy.Networks, policy.Assets)
if opt == nil { return errNoAcceptableOption }
if opt.Amount > policy.MaxPerRequest ||
    spentToday+opt.Amount > policy.DailyCap {
    return errOverBudget }
auth := signEIP3009(wallet, opt, ttl(opt.MaxTimeoutSeconds))
resp = get(url, "PAYMENT-SIGNATURE", encode(auth))
if resp.Status == 200 {
    spentToday += opt.Amount
    log(decode(resp.Header["PAYMENT-RESPONSE"]).Tx)
}
// paid retry returns 402 again → stop, do not sign again
```

UCP: where x402 fits in agentic commerce

- UCP: Universal Commerce Protocol, Google + Shopify, Jan 2026; Walmart, Target, ... on board
 - Standardizes the shopping journey for agents: GET /.well-known/ucp → capabilities, versions, JSON schemas, payment handlers
 - Product discovery: catalog search and product detail as REST endpoints
 - Shopping API = REST endpoints: POST /checkout-sessions, GET/PUT /checkout-sessions/{id}, POST .../complete (payment), POST .../cancel, GET /orders/{id}; also as JSON-RPC, MCP or A2A
- UCP and x402
 - Merchant lists payment handlers, e.g. card, x402
 - Agent picks one at /complete
- x402 as a UCP handler
 - Manifest lists handler "x402" with networks / assets
 - Checkout: agent calls complete → merchant answers 402 + PAYMENT-REQUIRED (amount = total, payTo = merchant)
 - Agent applies budget policy, signs, retries with PAYMENT-SIGNATURE
 - Merchant verifies + settles via facilitator → checkout "completed"
- Skeptical view
 - UCP is built for card rails and Google's identity; x402 is machine-to-machine API pay
 - [x402-in-UCP](#) is proof of concept, not standard