

OpenKitx403

v1.0 – PRODUCTION
GitHub Docs Quick Start
DOCUMENTATION

- Overview
- Features
- Why OpenKitx403
- How It Works
- Quick Start
- Installation
  - TypeScript Client
  - Python Server
  - Python Client
  - Agent SDK
- Client SDK
  - Browser Usage
  - Node.js Usage
- Server SDK
  - FastAPI Middleware
- Protocol Specification
- LangChain & Agents
- Security
- Internal Client Logic
- Internal Server Logic
- Complete Examples
- API Reference

OpenKitx403

HTTP-native wallet authentication for Solana

OpenKitx403 is an open-source, TypeScript-first protocol that standardizes **HTTP 403** as the semantic **"prove you control this wallet"** challenge for Solana wallet authentication.

Open Source MIT License Production Ready Type-Safe

Features

- **HTTP-native:** Uses standard HTTP 403 challenges, no custom protocols
- **Stateless:** No required server-side sessions or JWTs
- **Secure:** `ed25519-solana` signatures with replay protection
- **Production-ready:** Full TypeScript client + Python server/client SDKs
- **AI-friendly:** Agent SDK and LangChain tools
- **Token-gating ready:** Built-in support for NFT/token requirements
- **Multi-stack:** Browser, Node, FastAPI, and autonomous agents

Why OpenKitx403?

Traditional web authentication requires centralized identity providers or long-lived sessions. OpenKitx403 enables **stateless, cryptographic wallet-based authentication** using standard HTTP semantics. It works with Phantom, Backpack, and Solflare wallets and is designed to be implemented across multiple languages.

Archived copy · cryp2nova.com

Core Idea

Instead of "login with email/password", your API says:

```
403 Forbidden
WWW-Authenticate: OpenKitx403 realm="api-v1", version="1", challenge="<base64url>"
```

The client signs that challenge with a Solana wallet and re-sends the request with:

```
Authorization: OpenKitx403 addr="<wallet>", sig="<signature>", challenge="<base64url>", ts="<iso8601>", nonce="<nonce>", bind="<method:path>"
```

The server verifies the signature, enforces bindings, and grants access.

How It Works

OpenKitx403 follows a simple challenge-response authentication flow:

1

Client Request

Client attempts to access a protected resource.

```
GET /protected
```

↓
2

Server Challenge

Server responds with 403 and a challenge.

```
WWW-Authenticate: OpenKitx403 ...
```

↓
3

User Signs

Wallet signs the challenge using `ed25519-solana`.

```
Ed25519 signature
```

↓
4

Client Retry

Client retries with an `Authorization: OpenKitx403 ...` header.

↓
5

Success

Server verifies and returns `200 OK`.

No Sessions, No JWTs, No OAuth

OpenKitx403 is stateless by design. Each request is authenticated via a signed challenge, with replay protection and optional token-gating on top.

CRYP2NOVA

Archived copy · cryp2nova.com

Quick Start

Get running in 5 minutes.

1. Install Packages

```
# TypeScript client (browser/Node)
npm install @openkitx403/client

# Python server (FastAPI)
pip install openkitx403

# Python client (scripts/agents)
pip install openkitx403-client

# Agent SDK (TypeScript + Python)
npm install @openkitx403/agent
pip install openkitx403-agent
```

2. Minimal FastAPI Server

```
from fastapi import FastAPI, Depends
from openkitx403 import OpenKit403Middleware, require_openkitx403_user

app = FastAPI(title="My Wallet-Protected API")

app.add_middleware(
    OpenKit403Middleware,
    audience="https://api.example.com",
    issuer="my-api-v1",
    ttl_seconds=60,
    clock_skew_seconds=120,
    bind_method_path=True,
    origin_binding=False,
    ua_binding=False,
    replay_backend="memory",
)

@app.get("/protected")
async def protected(user = Depends(require_openkitx403_user)):
    return {
        "message": f"Authenticated as {user.address}",
        "wallet": user.address,
    }
```

3. Minimal Browser Client

```
import { OpenKit403Client } from "@openkitx403/client";

const client = new OpenKit403Client();

// Connect wallet
await client.connect("phantom");

// Authenticate
const response = await client.authenticate({
  resource: "https://api.example.com/protected",
  method: "GET",
});

if (response.ok) {
  const data = await response.json();
  console.log("✅ Authenticated as:", client.getAddress());
  console.log("Data:", data);
} else {
  console.error("❌ Authentication failed:", response.status);
}
```

Archived copy · cryp2nova.com

Installation

TypeScript Client

For browser and Node.js applications:

```
npm install @openkitx403/client
```

Supports:

- Browser (Phantom, Backpack, Solflare wallets)
- Node.js (with Solana keypair)
- React, Vue, Svelte, etc.

Python Server (FastAPI)

For FastAPI applications:

```
pip install openkitx403
# or
poetry add openkitx403
```

Includes:

- FastAPI middleware (`OpenKit403Middleware`)
- Replay protection store (memory backend)
- CORS helpers for `WWW-Authenticate`

Python Client

For Python scripts, agents and automation:

```
pip install openkitx403-client
# or
poetry add openkitx403-client
```

Includes:

- Full 403 → challenge → sign → retry flow
- Solana keypair support via `solders`
- Clean error types (`OpenKit403ClientError`)

Agent SDK

For AI agents and LangChain workflows:

```
npm install @openkitx403/agent
pip install openkitx403-agent
```

Includes:

- High-level `OpenKit403Agent` for autonomous flows
- LangChain tool integration
- Helpers for resource-based authentication

Client SDK

Browser Usage

Using OpenKitx403 in the browser with wallet extensions:

```
import { OpenKit403Client, detectWallets } from "@openkitx403/client";

const client = new OpenKit403Client();
```

CRYP2NOVA

Archived copy · cryp2nova.com

```
const wallets = await detectWallets();
console.log("Detected wallets:", wallets);

// 1. Connect a wallet
await client.connect("phantom");

// 2. Authenticate with API
const response = await client.authenticate({
  resource: "https://api.example.com/protected",
  method: "GET",
});

if (response.ok) {
  console.log("Authenticated as:", client.getAddress());
  const data = await response.json();
  console.log("API response:", data);
} else {
  console.error("Auth failed:", response.status);
}
```

React Example

```
import { useState } from "react";
import { OpenKit403Client } from "@openkitx403/client";

export function App() {
  const [client] = useState(() => new OpenKit403Client());
  const [address, setAddress] = useState<string | null>(null);

  const login = async () => {
    await client.connect("phantom");
    const response = await client.authenticate({
      resource: "https://api.example.com/user/profile",
    });

    if (response.ok) {
      setAddress(client.getAddress());
    }
  };

  return (
    <div>
      {!address ? (
        <button onClick={login}>Connect Wallet</button>
      ) : (
        <p>Connected: {address}</p>
      )}
    </div>
  );
}
```

Node.js Usage

For Node.js scripts you can either use the browser-compatible client with a custom wallet adapter, or use the Python client from automation. Example using Python client:

```
from solders.keypair import Keypair
from openkitx403_client import OpenKit403Client

keypair = Keypair()
print("Wallet:", keypair.pubkey())

client = OpenKit403Client(keypair)

response = client.authenticate(
    url="https://api.example.com/protected",
    method="GET",
)
```

CRYP2NOVA

Archived copy · cryp2nova.com

```
if response.ok:
    print(response.json())
```

Server SDK

FastAPI Middleware

```
from typing import Optional, List, Callable
from fastapi import FastAPI, Depends
from openkitx403 import (
    OpenKit403Middleware,
    require_openkitx403_user,
    get_openkitx403_user,
)

app = FastAPI()

app.add_middleware(
    OpenKit403Middleware,
    audience="https://api.example.com",
    issuer="my-api-v1",
    ttl_seconds=60,
    clock_skew_seconds=120,
    bind_method_path=True,
    origin_binding=False,
    ua_binding=False,
    replay_backend="memory",
    excluded_paths=["/health", "/docs"],
)

@app.get("/protected")
async def protected(user = Depends(require_openkitx403_user)):
    return {"address": user.address}

@app.get("/optional")
async def optional(user = Depends(get_openkitx403_user)):
    if user:
        return {"message": f"Authenticated as {user.address}"}
    return {"message": "Anonymous"}
```

Protocol Specification (Summary)

Scheme & Version

- **Scheme:** `OpenKitx403`
- **Version:** `1`
- **Algorithm:** `ed25519-solana`

WWW-Authenticate Header (Server → Client)

```
WWW-Authenticate: OpenKitx403 realm="api-v1", version="1", challenge="<base64url(challenge_json)>"
```

Challenge JSON

Field	Description
`v`	Protocol version (must be 1)
`alg`	Algorithm (must be `ed25519-solana`)
`aud`	Audience / expected API origin
`iss`	Issuer / server identifier
`nonce`	Random single-use nonce
`iat`	Issued-at timestamp (ISO 8601)
`exp`	Expiration timestamp
`method`	HTTP method (if method/path binding enabled)

Archived copy · cryp2nova.com

Field	Description
`path`	HTTP path (if method/path binding enabled)
`uaBind`	Require User-Agent binding
`originBind`	Require Origin binding
`ext`	Extension data for custom requirements

Signing String

```
OpenKitx403 Challenge

domain: https://api.example.com
server: api-v1
nonce: E2o6p0q0Zl5PBjXc
ts: 2025-11-05T10:30:00Z
method: GET
path: /protected

payload: {"v":1,"alg":"ed25519-solana","aud":"https://api.example.com",...}
```

This string is signed with Ed25519 using the wallet's private key.

LangChain & Agent SDK

TypeScript Agent

```
import { OpenKit403Agent } from "@openkitx403/agent";

const agent = new OpenKit403Agent({
  wallet: "phantom",
  autoConnect: true,
});

const result = await agent.execute({
  resource: "https://api.example.com/protected",
});

console.log(result);
```

LangChain (TypeScript)

```
import { SolanaWalletAuthTool } from "@openkitx403/agent";
import { initializeAgentExecutorWithOptions } from "langchain/agents";
import { ChatOpenAI } from "@langchain/openai";

const model = new ChatOpenAI({ temperature: 0 });
const tools = [new SolanaWalletAuthTool("phantom")];

const executor = await initializeAgentExecutorWithOptions(tools, model, {
  agentType: "openai-functions",
});

const result = await executor.invoke({
  input: "Connect my Phantom wallet and fetch my profile from https://api.example.com/profile",
});
```

Python Agent

```
from solders.keypair import Keypair
from openkitx403_agent import OpenKit403Agent, AgentExecuteOptions

keypair = Keypair()
agent = OpenKit403Agent(keypair, auto_connect=True)

result = await agent.execute(
    AgentExecuteOptions(
        resource="https://api.example.com/protected",
```

Archived copy · cryp2nova.com

```
        method="GET",
    )
)

print(result)
```

Security Model

OpenKitx403 was designed around strong security primitives that eliminate entire classes of vulnerabilities common in traditional auth.

1. Ed25519-Solana Signatures

The protocol uses **ed25519-solana** signatures, matching Solana's wallet cryptography exactly.

2. No Sessions, No Cookies

Authentication is stateless. No server sessions, no JWTs, no refresh tokens. The client signs each protected request.

3. Replay Protection

- Nonce stored in a replay store
- Reused nonce is rejected
- Expired challenges are rejected
- Method/path mismatch is rejected

4. Binding Rules

The server binds each signature to:

- HTTP method
- HTTP path
- (Optional) Origin
- (Optional) User-Agent

5. Canonical JSON + Base64url

Prevents ambiguity attacks and ensures cross-language compatibility.

6. Clock Skew Safety

SDKs enforce timestamp tolerance (default +/- 120 seconds).

7. Payload Integrity

Payloads inside challenges cannot be tampered with due to canonicalization and signature binding.

Security Notice

OpenKitx403 is for authentication, not message-level encryption. Always use HTTPS in production.

Internal Client Logic (TS + Python)

TS: Extract Challenge

```
private extractChallenge(headers: Headers): Challenge {
  const header = headers.get("WWW-Authenticate");
  if (!header) throw new Error("Challenge missing");
  return decodeChallenge(header);
}
```

TS: Canonical JSON

```
const payload = JSON.stringify(challenge, Object.keys(challenge).sort());
```

Archived copy · cryp2nova.com

Python: Canonical JSON

```
payload = json.dumps(challenge, sort_keys=True, separators=(",", ":"))
```

Signing (TS)

```
const message = new TextEncoder().encode(signingString);
const signature = await this.wallet.signMessage(message);
```

Signing (Python)

```
signature = keypair.sign_message(signing_string.encode())
```

Authorization Header

```
Authorization: OpenKitx403 addr="{addr}", sig="{sig}", challenge="{b64}", ts="{ts}", nonce="{nonce}", bind="{method}:{path}"
```

Internal Server Logic (Python)

```
auth_header = request.headers.get("Authorization")
if not auth_header:
    # Issue challenge
    header_value, _ = create_challenge(
        method=request.method,
        path=request.url.path,
        audience=self.config.audience,
        issuer=self.config.issuer,
        ttl_seconds=self.config.ttl_seconds,
        ua_binding=self.config.ua_binding,
        origin_binding=self.config.origin_binding,
    )
    # 403 + WWW-Authenticate
    ...

# Verify authorization
result = await verify_authorization(
    auth_header=auth_header,
    method=request.method,
    path=request.url.path,
    config=self.config,
)  # checks version, alg, audience, issuer, expiry, nonce, bindings, signature

if not result.ok:
    # return 401/403
    ...

# Attach user
request.state.openkitx403_user = {"address": result.address}
```

Complete Examples

Browser + FastAPI Full Stack

Frontend (React)

```
import { useState } from "react";
import { OpenKit403Client } from "@openkitx403/client";

export function App() {
  const [client] = useState(() => new OpenKit403Client());
  const [address, setAddress] = useState<string | null>(null);
  const [profile, setProfile] = useState<any | null>(null);

  const login = async () => {
    await client.connect("phantom");
    const response = await client.authenticate({
      resource: "http://localhost:8000/profile",
```

Archived copy · cryp2nova.com

```
      method: "GET",
    });

    if (response.ok) {
      setAddress(client.getAddress());
      const data = await response.json();
      setProfile(data);
    }
  };

  return (
    <div>
      {!address ? (
        <button onClick={login}>Connect Wallet</button>
      ) : (
        <div>
          <p>Wallet: {address}</p>
          <pre>{JSON.stringify(profile, null, 2)}</pre>
        </div>
      )}
    </div>
  );
}
```

Backend (FastAPI)

```
from fastapi import FastAPI, Depends
from fastapi.middleware.cors import CORSMiddleware
from openkitx403 import OpenKit403Middleware, require_openkitx403_user

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.add_middleware(
    OpenKit403Middleware,
    audience="http://localhost:3000",
    issuer="my-api",
    ttl_seconds=60,
)

@app.get("/profile")
async def profile(user = Depends(require_openkitx403_user)):
    return {
        "address": user.address,
        "username": f"User_{user.address[:6]}",
        "joinedAt": "2025-01-01T00:00:00Z",
    }
```

Python Script → Protected API

```
from solders.keypair import Keypair
from openkitx403_client import OpenKit403Client

keypair = Keypair()
client = OpenKit403Client(keypair)

res = client.authenticate("https://api.example.com/admin", method="POST", json_data={"force": True})
print(res.json())
```

API Reference (High-Level)

TypeScript Client

Archived copy · cryp2nova.com

class OpenKit403Client

```
constructor(walletOrKeypair?: WalletProvider | Keypair)
```

Initialize the client with a browser wallet or a Solana keypair (Node).

```
async connect(wallet: WalletProvider | string): Promise<void>
```

Connect to a Solana wallet (e.g. `"phantom"`, `"backpack"`).

```
async authenticate(resource: string | AuthRequest, options?: RequestInit): Promise<Response>
```

Performs complete 403 challenge flow: request → 403 → sign → retry with Authorization → final Response.

```
getAddress(): string
```

Returns the currently connected wallet address.

```
disconnect(): void
```

Disconnects the current wallet.

Python Server

class OpenKit403Middleware

```
OpenKit403Middleware(audience: str, issuer: str, ttl_seconds: int = 60, clock_skew_seconds: int = 120, bind_method_path: bool = False, origin_binding: bool = False, ua_binding: bool = False, replay_backend: str = "memory", token_gate: Optional[Callable[[str], bool]] = None, excluded_paths: Optional[list[str]] = None, allowed_origins: Optional[list[str]] = None)
```

FastAPI middleware that issues challenges and verifies signatures.

Python Client

class OpenKit403Client

```
__init__(self, keypair: Keypair)
```

Initialize client with a Solana keypair.

```
authenticate(url: str, method: str = "GET", headers: dict | None = None, data: dict | None = None, json_data: dict | None = None) -> requests.Response
```

Executes the full challenge → sign → retry loop.

Python Agent SDK

class OpenKit403Agent

```
__init__(self, keypair: Keypair, options: AgentAuthOptions | None = None)
async execute(self, options: AgentExecuteOptions) -> AgentAuthResult
```

OpenKitx403

HTTP-native wallet authentication for Solana. Stateless, cryptographic, production-ready.

Open source • MIT License

Docs & Spec

- Complete Docs (GitHub)
- Usage Examples
- Security Notes

Packages

Archived copy · cryp2nova.com

- @openkitx403/client
- @openkitx403/agent
- openkitx403 (server)
- openkitx403-client
- openkitx403-agent

Community

- Issues
- X (Twitter)
- security@openkitx403.dev

Archived copy · cryp2nova.com