**Como Conectar e Autenticar no WebSocket API da Binance com Chave Ed25519**

Aqui está o guia passo a passo para conectar ao WebSocket API da Binance e autenticar usando uma chave Ed25519 (como a sua de 44 bytes). Isso permite que você envie ordens ou acesse funcionalidades privadas depois de autenticado.

**Pré-requisitos**

* **Node.js:** Versão 18 ou superior (para suporte a crypto.subtle com Ed25519).
* **Biblioteca WebSocket:** Instale o pacote ws com npm install ws.
* **Chaves da Binance:** Uma chave pública (apiKey) e uma chave secreta Ed25519 (apiSecret) gerada no site da Binance (ex.: MC4CAQAwBQYDK2VwBCIEI...).

**Passo a Passo**

**Passo 1: Configurar o Ambiente**

* **O Que Fazer:** Crie um arquivo JavaScript (ex.: connect.js) e adicione o básico para usar WebSocket.
* **Como:**
  + Abra seu editor (ex.: VS Code).
  + Escreva isso no início do arquivo:

javascript

CollapseWrapCopy

const WebSocket = require('ws');

* + Isso importa a biblioteca ws para criar a conexão WebSocket.
* **Por Quê:** A Binance usa WebSocket para comunicação em tempo real, e ws é uma biblioteca simples para isso no Node.js.

**Passo 2: Definir Suas Chaves**

* **O Que Fazer:** Coloque sua chave pública (apiKey) e secreta (apiSecret) no código.
* **Como:**
  + Adicione essas linhas após o require:

javascript

CollapseWrapCopy

const apiKey = 'sua-chave-publica-aqui'; *// Ex.: 'nYZcNg7kBZfDAGuKpSOceL1h1YUMzSSwRGZut3KUy32KdpYPB0JKi1TV8liqvkVQ'*

const apiSecret = 'sua-chave-secreta-ed25519-aqui'; *// Ex.: 'MC4CAQAwBQYDK2VwBCIEICtYrvoheU+SozG67W4syoWymQ7Z2bTkgvd+IDypO3AM'*

* + Substitua pelos valores reais das suas chaves da Binance.
* **Por Quê:** A chave pública identifica você, e a chave secreta (Ed25519) é usada para assinar a requisição, provando que é você mesmo.

**Passo 3: Criar a Função de Assinatura**

* **O Que Fazer:** Faça uma função que assina uma mensagem com sua chave Ed25519 usando a Web Crypto API.
* **Como:**
  + Adicione essa função ao código:

javascript

CollapseWrapCopy

async function signMessage(message) {

*// Converte a chave secreta de base64 para um buffer binário*

const secretBuffer = Buffer.from(apiSecret, 'base64');

*// Importa a chave como Ed25519 para a Web Crypto API*

const key = await crypto.subtle.importKey(

'pkcs8', *// Formato padrão para chaves privadas Ed25519*

secretBuffer,

{ name: 'Ed25519' }, *// Especifica o algoritmo Ed25519*

false, *// Não exportável*

['sign'] *// Uso: assinar mensagens*

);

*// Converte a mensagem em bytes e assina*

const encoder = new TextEncoder();

const signature = await crypto.subtle.sign(

{ name: 'Ed25519' },

key,

encoder.encode(message)

);

*// Retorna a assinatura em base64*

return Buffer.from(signature).toString('base64');

}

* **Por Quê:**
  + A Binance exige uma assinatura Ed25519 para autenticar. Usamos crypto.subtle (parte do Node.js) porque ela suporta Ed25519 e aceita chaves como a sua (44 bytes em base64).
  + importKey transforma a chave secreta em um formato que o Node.js entende.
  + sign cria a assinatura, e base64 é o formato que a Binance aceita para session.logon.

**Passo 4: Conectar ao WebSocket**

* **O Que Fazer:** Crie a conexão com o servidor WebSocket da Binance.
* **Como:**
  + Adicione isso ao código:

javascript

CollapseWrapCopy

const ws = new WebSocket('wss://ws-api.binance.com:443/ws-api/v3');

* **Por Quê:** Esse é o endereço oficial do WebSocket API da Binance (versão 3), onde você autentica e envia ordens.

**Passo 5: Enviar a Requisição de Autenticação**

* **O Que Fazer:** Quando a conexão abrir, envie uma mensagem session.logon com a assinatura.
* **Como:**
  + Adicione este evento após criar o ws:

javascript

CollapseWrapCopy

ws.on('open', async () => {

console.log('Conectado ao WebSocket API!');

*// Cria os parâmetros de autenticação*

const timestamp = Date.now(); *// Hora atual em milissegundos*

const params = `apiKey=${apiKey}&timestamp=${timestamp}`; *// String a ser assinada*

*// Gera a assinatura*

const signature = await signMessage(params);

*// Monta a requisição session.logon*

const request = {

id: 1, *// Um número para identificar a resposta*

method: 'session.logon', *// Método de autenticação*

params: {

apiKey: apiKey,

timestamp: timestamp,

signature: signature

}

};

*// Envia a requisição*

ws.send(JSON.stringify(request));

});

* **Por Quê:**
  + session.logon autentica sua sessão usando sua chave Ed25519.
  + timestamp garante que a requisição é recente (a Binance rejeita se for muito antiga).
  + params é a string que você assina, seguindo o formato da documentação da Binance (apiKey=<key>&timestamp=<time>).
  + signature prova que você tem a chave secreta correspondente.

**Passo 6: Verificar a Resposta**

* **O Que Fazer:** Escute as mensagens do servidor para confirmar a autenticação.
* **Como:**
  + Adicione este evento após o ws.on('open'):

javascript

CollapseWrapCopy

ws.on('message', (data) => {

const msg = JSON.parse(data); *// Converte a resposta em objeto*

console.log('Mensagem recebida:', msg);

*// Checa se a autenticação deu certo*

if (msg.id === 1 && msg.status === 200) {

console.log('Autenticado com sucesso! Pronto para enviar ordens.');

}

});

* **Por Quê:**
  + A Binance responde com um objeto JSON. Se status for 200, você está autenticado.
  + id: 1 confirma que é a resposta à nossa requisição session.logon.

**Passo 7: Tratar Erros e Fechamento (Opcional)**

* **O Que Fazer:** Adicione eventos para erros e fechamento da conexão.
* **Como:**
  + Adicione isso ao final:

javascript

CollapseWrapCopy

ws.on('error', (error) => {

console.error('Erro:', error);

});

ws.on('close', () => {

console.log('Conexão fechada.');

});

* **Por Quê:** Isso ajuda a depurar problemas (ex.: chave inválida) e saber quando a conexão acaba.

**Código Completo**

Juntando tudo, fica assim:

javascript

CollapseWrapCopy

const WebSocket = require('ws');

const apiKey = 'sua-chave-publica-aqui';

const apiSecret = 'sua-chave-secreta-ed25519-aqui';

async function signMessage(message) {

const secretBuffer = Buffer.from(apiSecret, 'base64');

const key = await crypto.subtle.importKey(

'pkcs8',

secretBuffer,

{ name: 'Ed25519' },

false,

['sign']

);

const encoder = new TextEncoder();

const signature = await crypto.subtle.sign(

{ name: 'Ed25519' },

key,

encoder.encode(message)

);

return Buffer.from(signature).toString('base64');

}

const ws = new WebSocket('wss://ws-api.binance.com:443/ws-api/v3');

ws.on('open', async () => {

console.log('Conectado ao WebSocket API!');

const timestamp = Date.now();

const params = `apiKey=${apiKey}&timestamp=${timestamp}`;

const signature = await signMessage(params);

const request = {

id: 1,

method: 'session.logon',

params: {

apiKey: apiKey,

timestamp: timestamp,

signature: signature

}

};

ws.send(JSON.stringify(request));

});

ws.on('message', (data) => {

const msg = JSON.parse(data);

console.log('Mensagem recebida:', msg);

if (msg.id === 1 && msg.status === 200) {

console.log('Autenticado com sucesso! Pronto para enviar ordens.');

}

});

ws.on('error', (error) => {

console.error('Erro:', error);

});

ws.on('close', () => {

console.log('Conexão fechada.');

});

**Como Testar**

1. Salve o código como connect.js.
2. Instale a dependência:

bash

CollapseWrapCopy

npm install ws

1. Rode com Node.js:

bash

CollapseWrapCopy

node connect.js

1. Veja a saída no console:
   * "Conectado ao WebSocket API!"
   * "Mensagem recebida: { id: 1, status: 200, ... }"
   * "Autenticado com sucesso! Pronto para enviar ordens."

**Notas Finais**

* **Por Que Ed25519?** A Binance exige isso para o WebSocket API. Sua chave secreta de 44 bytes (em base64) é uma chave Ed25519 que a Web Crypto API consegue usar.
* **Problemas Comuns:**
  + Se der erro, cheque se as chaves estão certas e se o Node.js é 18+.
  + Um aviso "ExperimentalWarning" pode aparecer (Ed25519 é experimental até Node 23.5.0), mas não impede o funcionamento.