# Authentication

Authentication is crucial for ensuring that only authorized users can access private APIs. This document outlines the authentication mechanisms used for public and private APIs.

**Public API**

Public APIs do not require authentication. These interfaces are accessible to anyone without the need for any credentials.

Copy

No authentication is required for public interfaces.

**Private API**

Private APIs require authentication to ensure that only authorized users can access them. Authentication is achieved using custom headers that include a timestamp and a signature.

**Auth Header**

The following headers must be included in the request to authenticate access to private APIs:

Name

Location

Type

Required

Description

X-edgeX-Api-Timestamp

header

string

must

The timestamp when the request was made. This helps prevent replay attacks.

X-edgeX-Api-Signature

header

string

must

The signature generated using the private key and request details.

**CURL Examble**

Copy

curl --location --request GET 'https://pro.edgex.exchange/api/v1/private/account/getPositionTransactionPage?filterTypeList=SETTLE\_FUNDING\_FEE&size=10&accountId=544159487963955214' \--header 'X-edgeX-Api-Signature: 06d28020763542c0afc296dc8743797c6fda8ea9727745b57b671f70326dfed6077cd\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*aff3162e39d05d9df1c3ddf9648650382d6e62ff1076b14c0e6c687088d3917d8490e5412a080a6e9ea940c720ddd' \--header 'X-edgeX-Api-Timestamp: 1736313025024'

**Signature Elements**

The signature is generated using the following elements:

**Signature Element**

**Description**

X-edgeX-Api-Timestamp

The timestamp when the request was made. This is retrieved from the request header.

**Request Method (Uppercase)**

The HTTP method of the request, converted to uppercase (e.g., GET, POST).

**Request Path**

The URI path of the request (e.g., /api/v1/resource).

**Request Parameter/Body**

The query parameters or request body, sorted alphabetically.

**Request Parameter To Signature Content**

The request parameters are concatenated into a single string that forms the signature content. This string includes the timestamp, HTTP method, request path, and sorted query parameters or request body, ensuring the integrity and authenticity of the request.

For example, the following request parameters are concatenated into a single string:

1735542383256GET/api/v1/private/account/getPositionTransactionPageaccountId=543429922991899150&filterTypeList=SETTLE\_FUNDING\_FEE&size=10

**Generate Signature Java Example**

Below is a Java implementation of the Ecdsa signature algorithm. This example demonstrates how to sign a message using a private key.

**Private API Auth Signature:** This is used for authentication. We do not want the hash computation to consume excessive CPU resources. Therefore, this will use SHA3 to hash the request body string before signing.

Copy

import java.math.BigInteger;import org.web3j.abi.TypeEncoder;import org.web3j.abi.datatypes.Utf8String;import org.web3j.abi.datatypes.generated.Uint256;import org.web3j.crypto.Hash;import org.web3j.utils.Numeric;public class EcdsaSignatureDemo { public static final BigInteger K\_MODULUS = Numeric .toBigInt("0x0800000000000010ffffffffffffffffb781126dcae7b2321e66a241adc64d2f"); public static void main(String[] args) { String privateKeyHex = "0463ac809cc7d7c1baf\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*baff9fc6e3d8e5b160ea3fc"; // Ensure that the private key is a hexadecimal string without the "0x" prefix. if (privateKeyHex.startsWith("0x")) { privateKeyHex = privateKeyHex.substring(2); } BigInteger mySecretKey = new BigInteger(privateKeyHex, 16); PrivateKey privateKey = PrivateKey.create(mySecretKey); String message = "1735542383256GET/api/v1/private/account/getPositionTransactionPageaccountId=543429922991899150&filterTypeList=SETTLE\_FUNDING\_FEE&size=10"; String msg = TypeEncoder.encodePacked(new Utf8String(message)); BigInteger msgHash = Numeric.toBigInt(Hash.sha3(Numeric.hexStringToByteArray(msg))); msgHash = msgHash.mod(K\_MODULUS); Signature signature = Ecdsa.sign(msgHash, privateKey); String starkSignature = TypeEncoder.encodePacked(new Uint256(signature.r)) + TypeEncoder.encodePacked(new Uint256(signature.s)) + TypeEncoder.encodePacked(new Uint256(privateKey.publicKey().point.y)); System.out.println(starkSignature); } public static Signature sign(BigInteger msgHash, PrivateKey privateKey) { Curve curve = privateKey.curve; BigInteger randNum = new BigInteger(curve.N.toByteArray().length \* 8 - 1, new SecureRandom()).abs().add(BigInteger.ONE); Point randomSignPoint = EcMath.multiply(curve.G, randNum, curve.N, curve.A, curve.P); BigInteger r = randomSignPoint.x.mod(curve.N); BigInteger s = ((msgHash.add(r.multiply(privateKey.secret))).multiply(EcMath.inv(randNum, curve.N))).mod(curve.N); return Signature.create(r, s); }}

**Request Body To Body String Code Example**

The following Java code example demonstrates how to convert a JSON request body into a sorted string format suitable for signature generation:

Copy

import com.google.gson.JsonArray;import com.google.gson.JsonElement;import com.google.gson.JsonObject;import java.util.ArrayList;import java.util.List;import java.util.TreeMap;import java.util.stream.Collectors;public class RequestBodyToString { private static final String EMPTY\_STRING = ""; private static String getValue(JsonElement valueJson) { if (valueJson.isJsonNull()) { return EMPTY\_STRING; } else if (valueJson.isJsonPrimitive()) { return valueJson.getAsString(); } else if (valueJson.isJsonArray()) { JsonArray valueArray = valueJson.getAsJsonArray(); if (valueArray.isEmpty()) { return EMPTY\_STRING; } List<String> values = new ArrayList<>(); for (JsonElement itemValue : valueArray) { values.add(getValue(itemValue)); } return String.join("&", values); } else if (valueJson.isJsonObject()) { TreeMap<String, String> sortedDataMap = new TreeMap<>(); JsonObject valueJsonObj = valueJson.getAsJsonObject(); for (String key : valueJsonObj.keySet()) { sortedDataMap.put(key, getValue(valueJsonObj.get(key))); } return sortedDataMap.keySet().stream() .map(key -> key + "=" + sortedDataMap.get(key)) .collect(Collectors.joining("&")); } return EMPTY\_STRING; }}

**Signature Algorithm**

The signature algorithm used is Ecdsa (Elliptic Curve Digital Signature Algorithm).