OTPAuth Smart Contract

EtherScan

---

1. Introduction

The **OTPAuth Smart Contract** is designed to provide an on-chain solution for One-Time Password (OTP) authentication. This contract securely associates a username and public key with a seed and ensures OTP generation and validation based on the current timestamp.

2. Features

Core Functionalities

- **User Registration**:
  - Associates a username, public key, and OTP seed securely.
  - Prevents duplicate registration for usernames and public keys.
- **OTP Generation**:
  - Dynamically generates a time-based OTP using the user's seed and the current timestamp.
- **OTP Authentication**:
  - Validates the provided OTP and prevents reuse.
  - Ensures OTP expiration after the validity period.

3. Architecture

Key Components

1. **State Variables**:
   - usernameToDetails: Maps usernames to user details (publicKey, otpSeed, lastOTPTime).
   - publicKeyToUsername: Maps public keys to usernames for reverse lookup.
2. **Constants**:
   - OTP_VALIDITY_PERIOD: Sets the OTP validity duration (30 seconds).
3. **Structs**:
   - User: Contains user-specific information like publicKey, otpSeed, and lastOTPTime.

4. Smart Contract Details

State Variables

```
mapping(string => User) private usernameToDetails;
mapping(address => string) private publicKeyToUsername;
uint256 private constant OTP_VALIDITY_PERIOD = 30;
```

Events

- UserRegistered(string username, address publicKey): Emitted when a user registers successfully.
- OTPGenerated(string username, uint256 otp): (Optional) Emitted when an OTP is generated.
- OTPAuthenticated(address publicKey, bool success): Emitted after OTP authentication.

Functions

1. User Registration

function registerUser(string memory username, address publicKey, string memory otpSeed) public;

- **Description**: Registers a user with a username, public key, and OTP seed.
- **Reverts**:
  - If the username is already registered.
  - If the public key is already registered.
- **Event Emitted**: UserRegistered.

2. OTP Generation

function generateOTP(string memory username) public view returns (uint256);

- **Description**: Generates an OTP for the given username based on the current timestamp.
- **Reverts**: If the username is not found.
- **Logic**:
  - Uses keccak256 to hash the otpSeed with the current timestamp divided by OTP_VALIDITY_PERIOD.
  - Converts the hash to a 6-digit OTP (% 10**6).

3. OTP Authentication

function authenticate(address publicKey, uint256 otp) public;

- **Description**: Authenticates the provided OTP for a given public key.
- **Reverts**:
  - If the public key is not registered.
  - If the OTP is invalid or expired.
  - If the OTP was already used.
- **Event Emitted**: OTPAuthenticated.

4. Internal Function: _generateHash

function _generateHash(string memory seed, uint256 timestamp) private pure returns (bytes32);

- **Description**: Computes a hash using the user's seed and the current timestamp.
- **Returns**: bytes32 hash value.

5. Deployment

- **Prerequisites**:
  - Solidity compiler version ^0.8.0.
  - Deploy the contract on an Ethereum-compatible blockchain.
- **Steps**:
  1. Deploy the OTPAuth contract.
  2. Integrate the contract with off-chain services or a frontend for user registration and OTP generation/authentication.

6. Hardhat Testing

Test Cases

1. User Registration

- **Scenario**: Registers a user with a unique username and public key.
- **Expected Result**:
  - UserRegistered event is emitted.
  - Mapping entries (usernameToDetails, publicKeyToUsername) are updated.

2. OTP Generation

- **Scenario**: Generates a valid OTP for a registered user.
- **Expected Result**:

- Returns a 6-digit numeric OTP.

3. OTP Authentication

- **Scenario**: Authenticates a valid OTP for a user.
- **Expected Result**:
  - OTPAuthenticated event is emitted with success = true.

4. Invalid OTP

- **Scenario**: Attempts authentication with an incorrect OTP.
- **Expected Result**:
  - Reverts with "Invalid OTP or Expired".

5. OTP Reuse

- **Scenario**: Attempts to reuse an already authenticated OTP.
- **Expected Result**:
  - Reverts with "OTP already used".

6. Duplicate Registration

- **Scenario**: Attempts to register a duplicate username or public key.
- **Expected Result**:
  - Reverts with "Username already exists" or "Public key already registered".

Hardhat Test Code

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("OTPAuth Smart Contract", function () {
  let OTPAuth, otpAuth, user1, user2;

  beforeEach(async function () {
    OTPAuth = await ethers.getContractFactory("OTPAuth");
    otpAuth = await OTPAuth.deploy();
    [owner, user1, user2] = await ethers.getSigners();
  });

  it("should register a user successfully", async function () {
    const username = "user1";
    const seed = "seed123";
    const publicKey = user1.address;

    await expect(otpAuth.connect(user1).registerUser(username, publicKey,
seed))
```

```
            .to.emit(otpAuth, "UserRegistered")
            .withArgs(username, publicKey);
    });

    it("should generate a valid OTP for a user", async function () {
        const username = "user1";
        const seed = "seed123";
        await otpAuth.connect(user1).registerUser(username, user1.address,
seed);
        const otp = await otpAuth.generateOTP(username);
        expect(Number(otp)).to.be.a("number");
    });

    it("should authenticate a user with a valid OTP", async function () {
        const username = "user1";
        const seed = "seed123";
        await otpAuth.connect(user1).registerUser(username, user1.address,
seed);
        const otp = await otpAuth.generateOTP(username);
        await expect(otpAuth.authenticate(user1.address, otp))
            .to.emit(otpAuth, "OTPAuthenticated")
            .withArgs(user1.address, true);
    });

    it("should prevent OTP reuse", async function () {
        const username = "user1";
        const seed = "seed123";
        await otpAuth.connect(user1).registerUser(username, user1.address,
seed);
        const otp = await otpAuth.generateOTP(username);
        await otpAuth.authenticate(user1.address, otp);
        await expect(otpAuth.authenticate(user1.address,
otp)).to.be.revertedWith("OTP already used");
    });

    it("should not allow duplicate usernames or public keys", async function ()
{
        const username = "user1";
        const seed = "seed123";
        await otpAuth.connect(user1).registerUser(username, user1.address,
seed);
        await expect(otpAuth.connect(user2).registerUser(username,
user2.address, seed))
            .to.be.revertedWith("Username already exists");
    });
});
```

7. Security Considerations

- Prevents OTP reuse and ensures time-based validity.
- Requires secure storage of OTP seeds off-chain for enhanced security.
- Protects against duplicate username and public key registration.