L17: ERC-20 Token Standard

Module B: Ethereum & Smart Contracts

Blockchain & Cryptocurrency Course

December 2025

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the purpose and importance of the ERC-20 standard
- Describe the six required ERC-20 interface functions
- Understand the allowance mechanism for delegated transfers
- Implement a basic ERC-20 token from scratch
- Use OpenZeppelin's audited ERC-20 implementation
- Analyze real-world ERC-20 tokens (USDC, DAI, LINK)

What is ERC-20?

ERC-20 is the dominant fungible token standard on Ethereum:

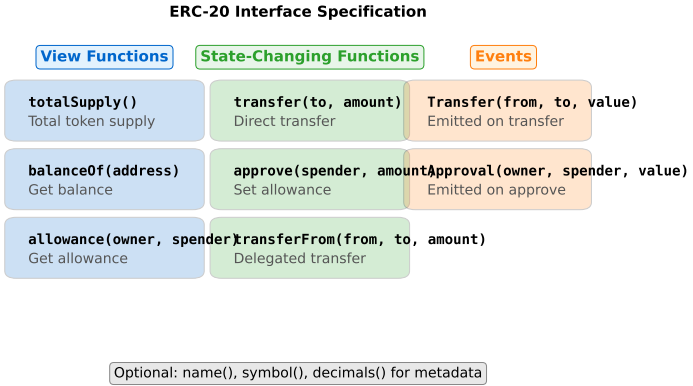
- **ERC:** Ethereum Request for Comments (proposal process)
- **Fungible:** Each token is identical and interchangeable
- **Standard:** Common interface enables interoperability

Why Standardization Matters:

- Wallets support all ERC-20 tokens without custom code
- Exchanges can list new tokens easily
- Smart contracts can interact with any ERC-20 token

ERC-20 Interface Overview

Six required functions + two required events:



ERC-20 Interface: Required Functions

```
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
        external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

Optional Metadata Functions

Not required but widely used:

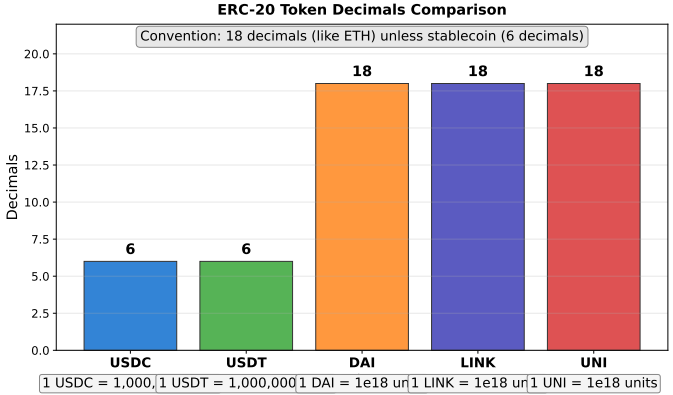
```
function name() public view returns (string memory);    // e.g., "US Dollar Coin"
function symbol() public view returns (string memory);  // e.g., "USDC"
function decimals() public view returns (uint8);        // e.g., 18 or 6
```

Decimals Explained:

- Tokens are stored as integers (no floating point on EVM)
- `decimals` defines how to display token amounts
- Example: 1 USDC = 1,000,000 units (6 decimals)
- Example: 1 DAI = 1e18 units (18 decimals)

Token Decimals Comparison

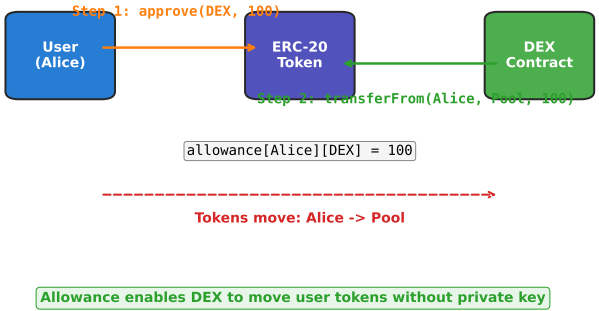
Different tokens use different decimal places:



Understanding Allowance Mechanism

Problem: How can a smart contract spend your tokens?
Solution: Two-step approve + transferFrom pattern

ERC-20 Allowance Mechanism



Allowance enables DEX to move user tokens without private key

Allowance Example: DEX Trading

Scenario: Alice wants to trade 100 DAI for ETH on Uniswap
Step-by-Step:

1. Alice calls `DAI.approve(UniswapRouter, 100e18)`
   - Sets allowance: Uniswap can spend 100 DAI
2. Alice calls `UniswapRouter.swapTokensForETH(100e18, ...)`
   - Uniswap calls `DAI.transferFrom(Alice, Pool, 100e18)`
   - Alice receives ETH in return

Infinite Approval:

- Users often approve `type(uint256).max` to avoid repeated approvals
- Risk: If DEX contract is hacked, all approved tokens can be stolen

Basic ERC-20 Implementation (Part 1)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract BasicERC20 {
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public totalSupply;

    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(string memory _name, string memory _symbol, uint8 _decimals,
                uint256 _initialSupply) {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        totalSupply = _initialSupply * 10**_decimals;
        balanceOf[msg.sender] = totalSupply;
        emit Transfer(address(0), msg.sender, totalSupply);
    }
```

Basic ERC-20 Implementation (Part 2)

```
    function transfer(address recipient, uint256 amount) public returns (bool) {
        require(recipient != address(0), "Transfer to zero address");
        require(balanceOf[msg.sender] >= amount, "Insufficient balance");
        balanceOf[msg.sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(msg.sender, recipient, amount);
        return true;
    }

    function approve(address spender, uint256 amount) public returns (bool) {
        require(spender != address(0), "Approve to zero address");
        allowance[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        require(balanceOf[sender] >= amount, "Insufficient balance");
        require(allowance[sender][msg.sender] >= amount, "Insufficient allowance");
        balanceOf[sender] -= amount;
        balanceOf[recipient] += amount;
        allowance[sender][msg.sender] -= amount;
        emit Transfer(sender, recipient, amount);
        return true;
    }
}
```

Security Considerations

Common ERC-20 Vulnerabilities:

1. **Approve Race Condition:**
   - Problem: Changing allowance from A to B allows spender to spend A+B
   - Solution: Use `increaseAllowance()` and `decreaseAllowance()`
2. **Integer Overflow (pre-0.8.0):**
   - Solution: Use Solidity 0.8+ (built-in overflow checks) or SafeMath
3. **Fee-on-Transfer Tokens:**
   - Some tokens deduct fees on transfer (e.g., SAFEMOON)
   - Breaks assumption that `transfer(amount)` sends exactly `amount`

OpenZeppelin ERC-20 Implementation

Industry-standard audited implementation:

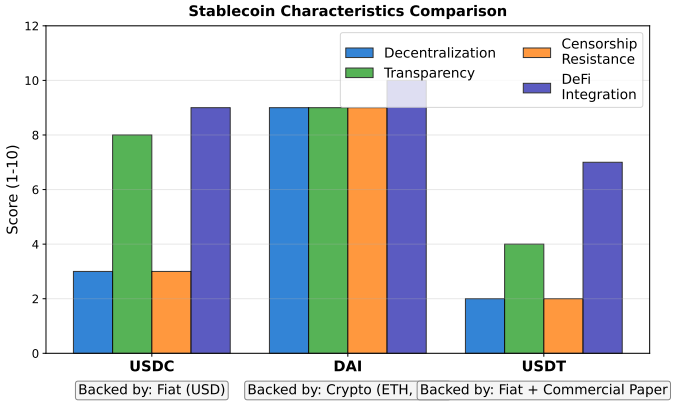
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("My Token", "MTK") {
        _mint(msg.sender, initialSupply * 10**decimals());
    }
}
```

Advantages:

- Battle-tested code (used by USDC, LINK, thousands of projects)
- Security audits by leading firms
- Extensions: ERC20Burnable, ERC20Pausable, ERC20Permit

Real-World Stablecoin Comparison

USDC vs DAI vs USDT characteristics:



Real-World Example: USDC

USD Coin (USDC) - Circle's stablecoin:
Key Features:

- Pegged 1:1 to US Dollar, backed by cash and Treasuries
- 6 decimals (not 18, to match fiat cents)
- Upgradable proxy pattern
- Blacklist function (regulatory compliance)
- Pausable (emergency circuit breaker)

Contract Address: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48

Real-World Example: DAI

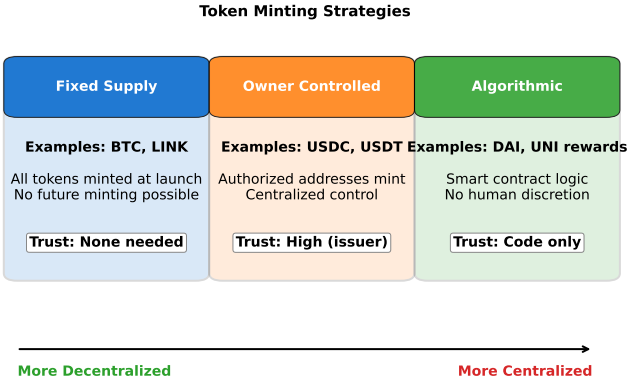
DAI - MakerDAO's decentralized stablecoin:
Key Features:

- Pegged to 1 USD via collateralized debt positions (CDPs)
- Over-collateralized by crypto assets (ETH, wBTC, USDC)
- 18 decimals (standard)
- No admin blacklist or pause (more decentralized than USDC)
- `permit()` - EIP-2612 gasless approvals

Contract Address: 0x6B175474E89094C44Da98b954EedeAC495271d0F

Token Minting Strategies

Different approaches to token supply management:



ERC-20 Variants and Improvements

Extensions to standard ERC-20:

1. **EIP-2612 (Permit):** Approve via off-chain signature
2. **ERC-20 Snapshot:** Capture balances at specific block for governance
3. **ERC-20 Votes:** Delegation of voting power (UNI, COMP)
4. **Rebase Tokens:** Balance changes automatically (Ampleforth)
5. **Wrapped Tokens:** ERC-20 wrapper around native assets (WETH)

Key Takeaways

1. **ERC-20 Standard:** Common interface for fungible tokens enabling interoperability
2. **Core Functions:** totalSupply, balanceOf, transfer, approve, allowance, transferFrom
3. **Allowance Mechanism:** Two-step approve + transferFrom for smart contract spending
4. **Security:** Use OpenZeppelin, beware of approve race condition
5. **Decimals:** Convention is 18, stablecoins often use 6
6. **Minting:** Fixed supply, owner-controlled, or algorithmic approaches

Discussion Questions

1. Why do stablecoins like USDC use 6 decimals instead of 18?
2. What are the security tradeoffs between limited vs infinite approval?
3. How does EIP-2612 permit() improve user experience?
4. What are the regulatory implications of USDC's blacklist function?
5. How might fee-on-transfer tokens break DEX contracts?

Next Lesson Preview: L18 - ERC-721 and ERC-1155 Standards

Coming up next:

- Non-fungible tokens (NFTs) and the ERC-721 standard
- ownerOf, tokenURI, safeTransferFrom functions
- Metadata standards and IPFS integration
- ERC-1155 multi-token standard (fungible + non-fungible)
- Real-world examples: CryptoPunks, Bored Apes, ENS domains