Smart Contracts and Blockchain Technology

Lecture 12. Token Programming

Christian Ewerhart

University of Zurich

Fall 2022

Introduction and overview

Last lecture

- Advanced Solidity language concepts

This lecture

- Tokens
- ERC20 standard
- NFTs

Token programming (1)

What is a token

A **token** is a privately issued special-purpose coin-like items of insignificant intrinsic value, such as a laundry token:

Blockchain tokens are conceptualy identical, but differ from laundry tokens in several ways:

- flexibility
- global scale
- transaction costs
- transparency

Token programming (2)

Token classes

Payment tokens:

- Cryptocurrencies (i.e., **native tokens** or **coins** embodied in blockchain systems)[1]
- Tokens representing cryptocurrencies on other chains

Asset tokens:

- Stablecoins, equity tokens, and DOA shares
- Digital collectibles (e.g., NFTs)

Utility tokens:

- Resource tokens (e.g., for CPU time)
- Certificates (e.g., access rights, identity tokens,...)

[1] These "tokens" do not rely on smart contract functionality and are intrinsic to the underlying blockchain system (e.g., bitcoin, ether).

Token programming (3)

Non-fungible tokens

Securities and tokens are called **fungible** when users consider individual units as perfect substitutes.[2]

Non-fungible tokens (NFTs) correspond (in a somewhat vage way) to a unique tangible or intangible item (e.g., a painting). Each non-fungible token is associated with a unique identifier, such as a serial number.

[2] Banknotes, e.g., are fungible, even though there are limitations (serial numbers and age, for instance).

Token programming (4)

Legal nature of tokens

Blockchain enthusists often refer to the ability of blockchain systems to convert extrinsic assets into intrinsic assets. However, developments are rather slow.[3]

- A token or NFT typically does not embody any legal claim.[4]
- Tokens may be rather illiquid, i.e., it may be hard to sell them on short notice at a good price.

Note: Tokens are not risky because of counterparty risk (i.e., the risk that a legal obligation is not fulfilled by a counterparty), it is the "triple-L" risk (i.e., legal nature, issuer liability, and liquidity are all questionable).

[3] An exception are equity tokens under Swiss law.

[4] E.g., virtual items in a computer game may not be considered as assets (here). This need not stop tax authorities to impose wealth tax on cryptoassets though.

Token programming (5)

Duck test

During the initial ICO hype around 2017, scammers have often tried to hide the fact that they were actually offering (worthless) securities.[5] To explain why this will not stop prosecution, regulators have referred to the so-called **duck test**:

"If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."

[5] An **Initial Coin Offering (ICO)** is a crowdfunding mechanism used to raise money by selling tokens. Unlike public offerings in the highly regulated securities markets, ICOs used to be essentially unregulated.

Token programming (6)

Who needs utility tokens?

Opportunities: Token issuers inherit the market enthusiasm, early adopters, technology, innovation, and liquidity of the entire token economy.

Challenges: Utility tokens introduce unnecessary costs/risks and adoption barriers for startups.

Token programming (7)

Tokens on Ethereum

Sending ether is an intrinsic action of the Ethereum platform, but sending or even owning tokens is not:

- The ether balance of Ethereum accounts is handled **at the protocol level**, whereas the token balance of Ethereum accounts is handled **at the smart contract level**.

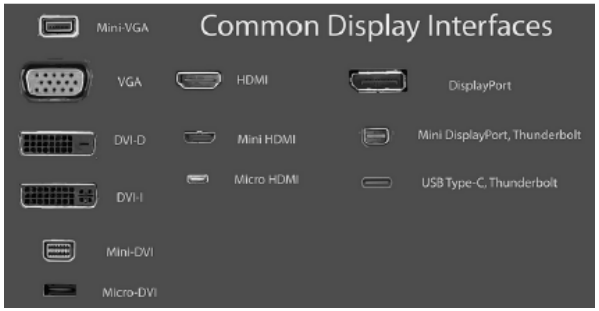
In order to create a new token on Ethereum, you must create a new smart contract that handles everything, including:

- ownership
- transfers, and
- access rights.

Token programming (8)

Standards

Standards help interoperability...



In a smart contract, token standards are the **minimum specifications** for an implementation (usually functions and events).[6]

Token contract that follow the standards may easily interact with wallets, exchanges, and existing user interfaces.

[6] How the minimum specifications are implemented does not matter. Moreover, additional functionality may be added ad libitum.

Token programming (9)

ERC20 required functions

The vast majority of tokens are currently based on the ERC20 standard.

Single-step work-flow:

- **totalSupply:** Returns the total units of this token that currently exist.[7]
- **balanceOf:** Given an address, returns the token balance of that address.
- **transfer:** Given an address and amount, transfers that amount of tokens to that address, from the balance of the address that executed the transfer.

[7] ERC20 tokens can have a fixed or a variable supply.

Token programming (10)

ERC20 required functions (continued)

Two-step work-flow:

- **transferFrom:** Given a sender, recipient, and amount, transfers tokens from one account to another.
- **approve:** Given a recipient address and amount, authorizes that address to execute several transfers up to that amount, from the account that issued the approval.
- **allowance:** Given an owner address and a spender address, returns the remaining amount that the spender is approved to withdraw from the owner.

Token programming (11)

ERC20 required events

Transfer. Event triggered upon a successful transfer (call to either transfer or transferFrom).[8]

Approval. Event logged upon a successful call to approve.

[8] Even for zero-value transfers.

Token programming (12)

ERC20 optional functions

In addition to the required functions listed in the previous section, the following optional functions are also defined by the standard:

- **name:** Returns the human-readable name (e.g., "US Dollars") of the token.
- **symbol:** Returns a human-readable symbol (e.g., "USD") for the token.
- **decimals:** Returns the number of decimals used to divide token amounts.[9]

[9] For example, if decimals is 2, then the token amount is divided by 100 to get its user representation.

Token programming (13)

ERC20 interface

```
contract ERC20 {
    function totalSupply() constant returns (uint theTotalSupply);
    function balanceOf(address _owner) constant returns (uint balance);
    function transfer(address _to, uint _value) returns (bool success);
    function transferFrom(address _from, address _to, uint _value) returns
        (bool success);
    function approve(address _spender, uint _value) returns (bool success);
    function allowance(address _owner, address _spender) constant returns
        (uint remaining);
    event Transfer(address indexed _from, address indexed _to, uint _value);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
}
```

Token programming (14)

ERC20 data structures

```
mapping(address => uint256) balances;
```

This implements an internal table of token balances, by owner, and allows the token contract to keep track of who owns the tokens. Each transfer is a deduction from one balance and an addition to another balance.[10]

[10] If Alice wants to send 10 tokens to Bob, her wallet sends a transaction to the token contract's address, calling the transfer function with Bob's address and 10 as the arguments. The token contract adjusts Alice's balance (–10) and Bob's balance (+10) and issues a Transfer event.

Token programming (15)

ERC20 data structures (continued)

```
mapping (address => mapping (address => uint256)) public allowed;
```

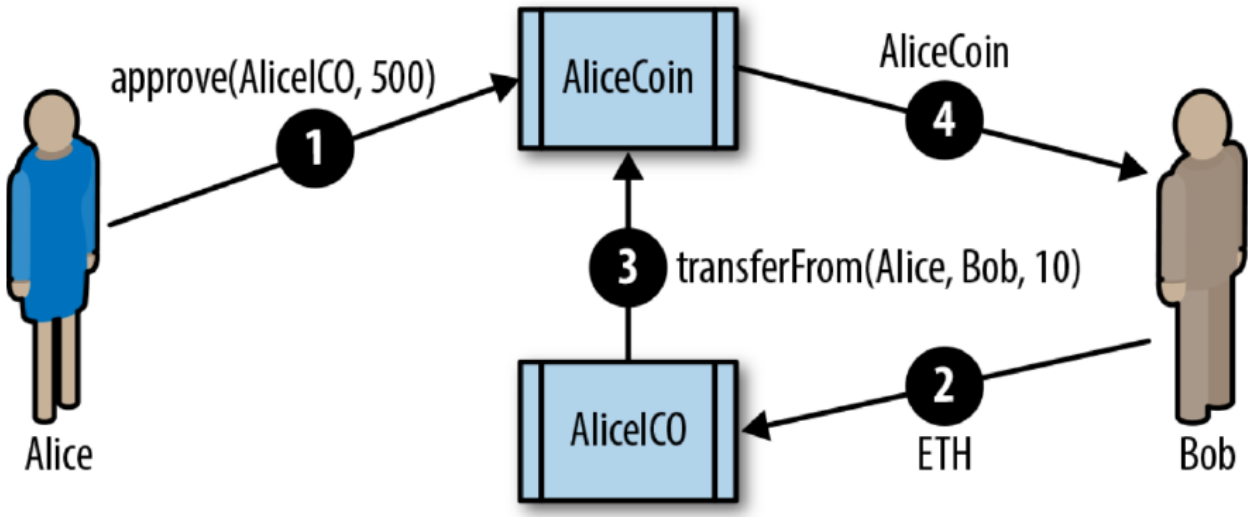
The owner of a token can delegate authority to a spender, allowing them to spend a specific amount (allowance) from the owner's balance.

The ERC20 contract keeps track of the allowances with a two-dimensional mapping, with the primary key being the address of the token owner, mapping to a spender address and an allowance amount.

Token programming (16)

The two-step workflow

Illustration. To sell tokens for an ICO, Alice approves the AliceICO crowdsale contract address to distribute a certain amount of tokens. The crowdsale contract can then `transferFrom` the token contract owner's balance to Bob, the buyer of the token:



Token programming (17)

ERC20 implementations

Consensys EIP20. A simple and easy-to-read implementation of an ERC20-compatible token.

OpenZeppelin StandardToken. This implementation is ERC20-compatible, with additional security precautions. It forms the basis of OpenZeppelin libraries implementing more complex ERC20-compatible tokens with fundraising caps, auctions, vesting schedules, and other features.

Token programming (18)

Truffle and Ganache

Truffle. Comprehensive suite of tools for smart contract development.

Ganache. Fires up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.

Token programming (19)

Warnings

You cannot pay for a transaction's gas with a token and the token contract can't pay the gas for you.

Example 1. When trying to send a token, your wallet informs you that you need ether to pay the **gas**.

Example 2. Polygon MATIC exists both as a native token and and ERC20 token, and the transfer is accomplished by a **plasma bridge**.

Token programming (20)

ERC721: Non-fungible Token Standard

The ERC721 proposal is for a standard for non-fungible tokens (a.k.a. **deeds**).

The use of the word "deed" is intended to reflect the "ownership of property" part, even though these are not recognized as legal documents in any jurisdiction (yet):[11]

- **virtual item**, such as an in-game item or digital collectible
- **physical item** whose ownership is tracked by a token (a house, a car, or an artwork).
- **items of negative value**, such as loans (debt), liens, easements, etc.

[11] A **deed** is a legal document that is signed and delivered, especially one regarding the ownership of property or legal rights.

Token programming (21)

ERC20 vs. ERC721

Look at the internal data structure used in ERC721:

```
// Mapping from deed ID to owner
mapping (uint256 => address) private deedOwner;
```

Whereas ERC20 tracks the balances that belong to each owner, with the owner being the primary key of the mapping, ERC721 tracks each deed ID and who owns it, with the deed ID being the primary key of the mapping.

Token programming (22)

ERC721 interface

```
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to, uint256 _deedId);
    event Approval(address indexed _owner, address indexed _approved,
                   uint256 _deedId);
    event ApprovalForAll(address indexed _owner, address indexed _operator,
                         bool _approved);

    function balanceOf(address _owner) external view returns (uint256 _balance);
    function ownerOf(uint256 _deedId) external view returns (address _owner);
    function transfer(address _to, uint256 _deedId) external payable;
    function transferFrom(address _from, address _to, uint256 _deedId)
        external payable;
    function approve(address _approved, uint256 _deedId) external payable;
    function setApprovalForAll(address _operator, boolean _approved) payable;
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

Token programming (23)

Common extensions to token standards

- **Owner control.** The ability to give specific addresses, or sets of addresses (i.e., multisignature schemes), special capabilities, such as minting, recovery, etc.
- **"Monetary policy."** The ability to add to or reduce the total supply of tokens, at a predictable rate or by "fiat" of the creator of the token. This also includes caps on supply.
- **Crowdfunding.** The ability to offer tokens for sale, for example through an auction or market sale.
- **Recovery backdoors.** Functions to recover funds, reverse transfers, or dismantle the token that can be activated by a designated address or set of addresses.
- **White-/blacklisting.** The ability to restrict actions (such as token transfers) to specific addresses. There is usually a mechanism for updating the lists.

Token programming (24)

Final words on tokens and ICOs

Tokens have been an explosive development in the Ethereum ecosystem.

Nevertheless, the importance and future impact of these standards should not be confused with an endorsement of past and current token offerings. Many of the tokens on offer in Ethereum today are barely disguised **scams, pyramid schemes, and money grabs**.

Separate this from the long-term vision and impact of smart contract technology.

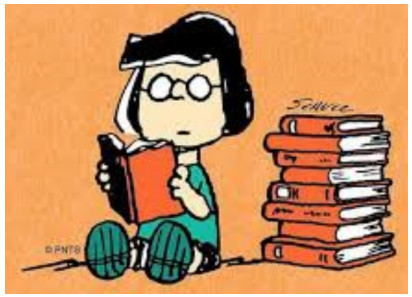
Token programming (25)

Bibliographic notes

This slide deck is based on Chapter 10 of Antonopoulos and Wood (2022). (link)

An insightful discussion of money, tokens, and other means of payment can be found in the classic monograph by Fisher (1911). (link)

The classification of tokens used above has been proposed by the Swiss financial supervisory authority (Finma). The ERC20 standard was introduced in November 2015 by Fabian Vogelsteller as an Ethereum Request for Comments (ERC).



Control structures in Solidity (29)

References

Antonopoulos and Wood (2022), Mastering Ethereum.

Fisher, I. (1911). The Purchasing Power of Money.