BDA project documentation

Modified ERC20 Token

Author:
Tomáš Beránek (xberan46)

Demonstration video

E-mail: **xberan46@stud.fit.vutbr.cz**
Date: **May 14th 2023**

1 ERC20 Token

ERC20 [3, 4] is a technical standard that defines a set of rules and requirements for fungible[1] tokens on the Ethereum blockchain. It is used to ensure interoperability between tokens and allows them to interact with each other seamlessly. ERC20 tokens are programmable digital assets that can represent various things such as transferable units of value, voting rights, or other functionalities.

The ERC20 standard includes 6 mandatory functions (and 2 events) that all ERC20 tokens must implement:

- `totalSupply()`

  returns the total number of tokens in circulation,

- `balanceOf(address owner)`

  returns the balance of `owner` address,

- `transfer(address to, uint256 value)`

  transfers `value` tokens from the sender's address to address `to`,

- `approve(address spender, uint256 value)`

  allows the `spender` address to spend up to `value` tokens on behalf of the sender,

- `allowance(address owner, address spender)`

  returns the number of tokens that the `spender` address is allowed to spend on behalf of the `owner` address,

- `transferFrom(address from, address to, uint256 value)`

  allows the sender to transfer `value` tokens from address `from` to address `to`, if the sender is authorized to spend the given amount.

These functions allow for the transfer of tokens between addresses, checking an account's balance, and approving third-party accounts to spend tokens on behalf of the token holder. By following these standards, ERC20 tokens can be easily integrated into decentralized applications (DAPPs) and other projects. They provide a standardized and interoperable way to represent digital assets on the Ethereum blockchain, making it easier to create decentralized applications and exchange value in a transparent and secure manner.

[1] **fungible tokens** are interchangeable with other tokens of the same type.

2 Modified ERC20 Token (BDAERC20)

The goal of this project is to implement a modified ERC20 token (hereafter referred to as BDAERC20). A list of all modifications is given in the file `assignment.pdf`. BDAERC20 is implemented in Solidity 0.8.13 and uses the Truffle framework for compilation, testing, deployment, etc.

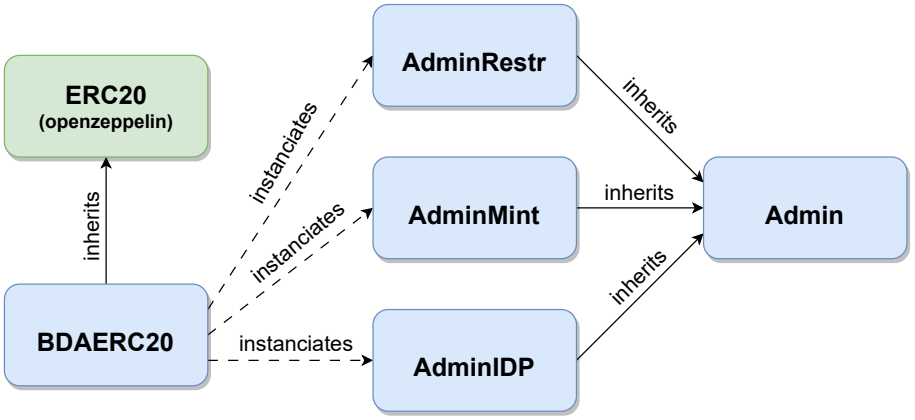

Figure 1: Graph of inheritance of implemented contracts. Blue are contracts that have been implemented in the project and green are existing contracts.

The implementation consists of 5 contracts, see Figure 1. The interface for communication with users is the `BDAERC20` contract, which instantiates contracts for individual admin roles – `AdminRestr`, `AdminMint` and `AdminIDP`. These admin contracts all inherit from the `Admin` contract, which provides basic admin account management (store admin addresses, add, remove etc.). `AdminRestr`, `AdminMint` and `AdminIDP` implement admin specific actions. The `BDAERC20` contract inherits from OpenZeppelin's `ERC20` [1] contract, and the `EnumerableSet` and `EnumerableMap` libraries also from OpenZeppelin are used as well. In addition to the mandatory and some optional ERC20 functions (see Chapter 1), `BDAERC20` provides the following functions:

- `isVerified(address user)`

  returns information about whether `user` is verified (returns `false` when `user` is revoked),

- `isRevoked(address user)`

  returns information about whether `user` is revoked,

- `isVerifiedUntil(address user)`

returns to which epoch time `user` is verified,

- `getVerifiedUsers()`

  returns a list of verified users,

- `transferedToday(address user)`

  returns the number of tokens sent today by the `user`,

- `mint(address[] accounts, uint256[] amounts)`

  mint `accounts` their respective number of `amounts` tokens,

- `signMint(address[] accounts, uint256[] amounts)`

  same as `mint`, but requires consensus of mint admins and does not count into `TMAX`,

- `signTransferLimitChange(address user, uint256 newValue)`

  after restriction admin consensus change daily transfer limit of `user` to `newValue`,

- `getTransferLimit(address user)`

  returns the daily transfer limit of `user`,

- `signTMAXChange(uint newTMAX)`

  after mint admin consensus change `TMAX` to `newTMAX`,

- `verify(bytes32 hashedMessage, uint8 v, bytes32 r, bytes32 s)`

  verifies the sender if the signature is valid and from a valid IDP,

- `getMintContractAddress()`

  returns the address of the `AdminMint` contract (for development purposes),

- `getIDPContractAddress()`

  returns the address of the `AdminIDP` contract (for development purposes),

- `signAddingMintingAdmin(address addr)`

  add `addr` as mint admin after mint admin consensus,

- `signAddingIDPAdmin(address addr)`

  add `addr` as IDP admin after IDP admin consensus,

- `signAddingRestrAdmin(address addr)`

  add `addr` as restriction admin after restriction admin consensus,
- `signRemovingMintingAdmin(address addr)`

  removes `addr` as mint admin after mint admin consensus,
- `signRemovingIDPAdmin(address addr)`

  remove `addr` as IDP admin after IDP admin consensus,
- `signRemovingRestrAdmin(address addr)`

  remove `addr` as restriction admin after restriction admins consensus,
- `getMintAdmins()`

  returns a list of mint admins,
- `getIDPAdmins()`

  returns a list of IDP admins,
- `getRestrAdmins()`

  returns a list of restriction admins,
- `isMintAdmin(address addr)`

  returns information about whether `addr` is a mint admin,
- `isIDPAdmin(address addr)`

  returns information about whether `addr` is an IDP admin,,
- `isRestrAdmin(address addr)`

  returns information about whether `addr` is a restriction admin,
- `isIDP(address addr)`

  returns information about whether `addr` is an IDP,
- `signAddingIDP(address addr)`

  add `addr` as IDP after IDP admin consensus,
- `signRemovingIDP(address addr)`

  removes `addr` as IDP after IDP admin consensus,
- `signRevoke(address addr)`

after IDP admin consensus revokes `addr`,

- `signApprove(address addr)`

  after IDP admin consensus approves back the `addr`,

- `getIDPs()`

  returns a list of IDPs,

- `mintedToday(address addr)`

  returns the number of tokens minted today by the `addr` mint admin,

- `getTMAX()`

  returns the `TMAX` value (how much mint admin can mint per day).

3 Decentralized application (DAPP)

For the purpose of accessing and controlling the BDAERC20 smart contract, a DAPP has been created to simplify access to the mentioned functions for users. The DAPP was created using a combination of JS, HTML and CSS (with the Bulma [2] CSS library). It is a web application that runs in a web browser (tested on Mozilla Firefox 112.0.2 (64-bit)) and uses the MetaMask wallet (tested on version 10.29.0) to provide access to the blockchain. A demonstration of the verified user UI can be seen in Figure 2. A detailed demonstration of DAPP is in the linked video[2] (in Czech).

4 Gas Measurements

The measurement of the gas usage of each function was performed directly in the Truffle tests of the BDAERC20 contract. In the following table, only gas consuming (state modifying) functions are shown. It should also be emphasized that the gas consumption of some functions is strongly dependent on the input parameters. For example, the `mint` function consumes 126k of gas when minted to a single user, but if minted to 3 users, the gas consumption increases to 194k. The measurement was designed to try to take into account the hardest path in the code (potentially the highest gas usage). Table 1 shows the gas consumption for individual functions.

[2] **DAPP demonstration video** – link.

Table 1: Gas usage for each (non view/pure) function in the BDAERC20 contract.

Function	Gas Used
signAddingIDP	111111
verify	159954
mint	194370
signMint	111357
transfer	89804
approve	55343
transferFrom	70970
signTransferLimitChange	136671
signTMAXChange	54625
signAddingMintingAdmin	96144
signAddingIDPAdmin	96189
signAddingRestrAdmin	96120
signRemovingMintingAdmin	67757
signRemovingIDPAdmin	67812
signRemovingRestrAdmin	67767
signRevoke	70118
signApprove	55751
signRemovingIDP	62035

5 Installation instructions for Ubuntu 20.04

This project (smart contracts and DAPP) was built for and tested on a local blockchain emulated by **ganache-cli** and managed by **Truffle** framework (compilation, testing and deployment of smart contracts) and **MetaMask** wallet (DAPP).

To install the **Truffle** framework, run:

```
sudo apt-get install -y nodejs build-essential
```

```
sudo npm install -g truffle
```

To install **ganache-cli**, run:

```
sudo npm install ganache-cli
```

Next, install the necessary smart contract dependencies using:

```
npm install
```

Install the **MetaMask** add-on in your browser (tested in Mozilla Firefox 112.0.2 (64-bit) with MetaMask 10.29.0).

Since DAPP needs the address of the deployed contract, this address is hardcoded at the beginning of the `dapp/script.js` file. In real world conditions, the address of the contract does not change, but when the local blockchain is restarted, the address will change unless **mnemonic** is specified. Mnemonic is already set inside `Makefile`. To start the local blockchain, run:

```
make blockchain
```

To deploy the BDAERC20 smart contract, run:

```
make deploy
```

To run DAPP using the `http.server` Python module, run:

```
make run-dapp
```

DAPP is now available through a web browser at http://localhost:8000.

To run Truffle tests and/or gas measurements, run:

```
truffle test
```

References

[1] *Contracts* [online]. [cit. 2023-05-14]. Dostupné z: https://docs.openzeppelin.com/contracts/4.x/.

[2] Jeremy Thomas. *Bulma Documentation* [online]. [cit. 2023-05-14]. Dostupné z: https://bulma.io/documentation/overview/.

[3] McKie, S. *The Anatomy of ERC20* [online]. BlockChannel, září 2017 [cit. 2023-05-14]. Dostupné z: https://medium.com/blockchannel/the-anatomy-of-erc20-c9e5c5ff1d02.

[4] Smith, Corwin and Ponnan, P. Jithil et al.. *ERC-20 Token Standard* [online]. Duben 2023 [cit. 2023-05-14]. Dostupné z: https://ethereum.org/cs/developers/docs/standards/tokens/erc-20/.

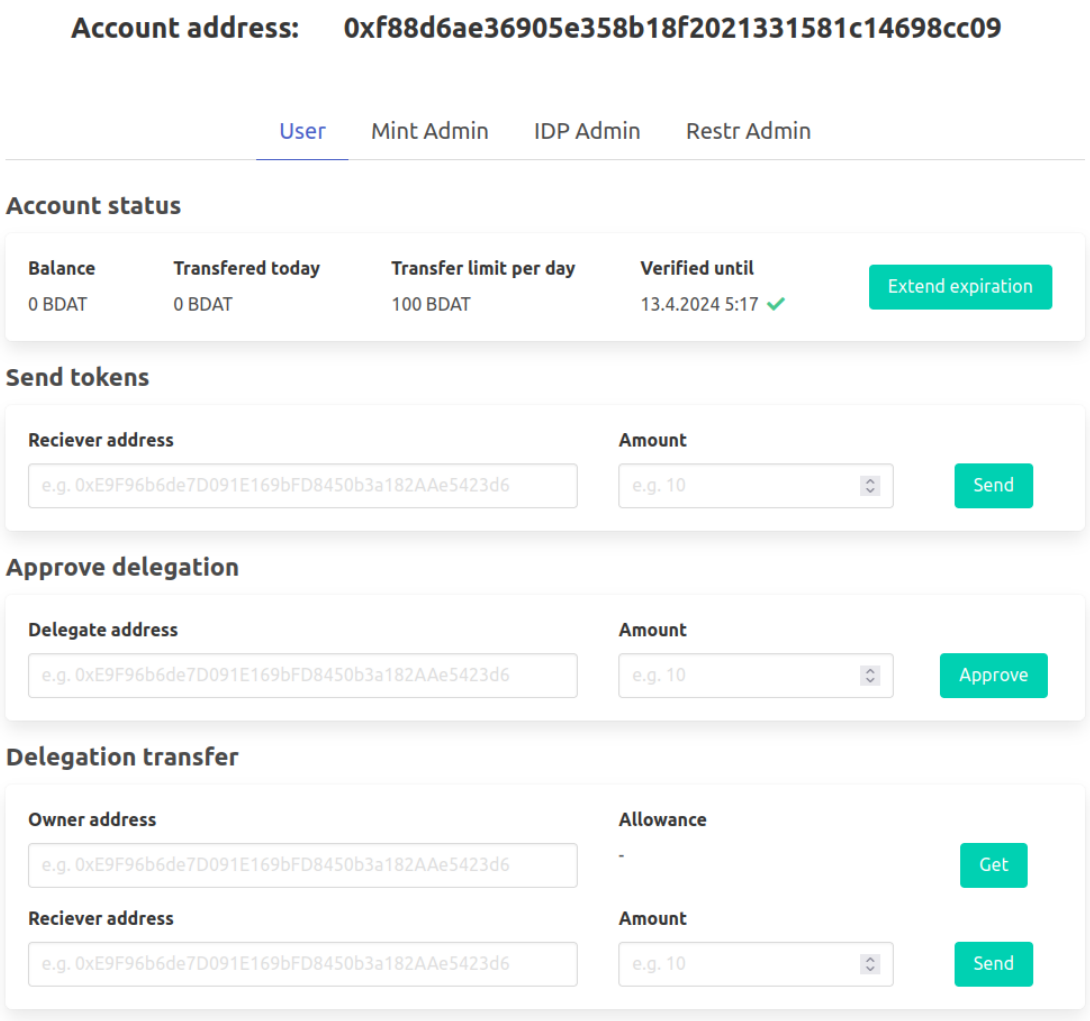

Figure 2: Screenshot of the verified user UI. The user can monitor the status of his account, send tokens, approve delegation, perform delegation, etc.