AlphaSwap Tokens

Introduction

The AlphaSwap token program is an extension of the credits.aleo / ARC20 by AlphaSwap, based on the characteristics of the aleo blockchain and the needs of the AlphaSwap functionality.

The main difference between AlphaSwap and credits.aleo/ARC20 programs is that AlphaSwap is a multi-token program, while credits.aleo and ARC20 are both single-token programs. This means that each credits.aleo / ARC20 token has its own unique program, while all AlphaSwap tokens are managed by the single AleoSwap program.

The AlphaSwap program manages the following types of tokens:

1. tokens created by calling AlphaSwap's `create_token` function
2. the pair tokens created by calling AlphaSwap's `create_pair` or `create_pair_privately` functions.
3. tokens wrapped into AlphaSwap by wrpper programs (e.g. WALEO is the wrapped token of aleo credits)

All the three types of tokens are handled in the same way within the AlphaSwap program.

Each token is assigned a unique token_id, which is used in all queries and function calls to tell the program which token is being queried or operated.

The latest AlphaSwap program is alphaswap_v1.aleo

Common Tokens

Here are some common tokens of alphaswap_v1.aleo on testnet3:

- USDT: 725661112884578732791551467370687855499176489412483327103550882637097088086 5field
    - decimals: 6
    - USDT is a token created by AlphaSwap for testing purposes.
    - It can be claimed freely from the AlphaSwap faucet.
- ALS: 256714891143338667748530468478598388103520303958976170022647522627234654226field
    - decimals: 6
    - This ALS is a token created by AlphaSwap for testing purposes.
    - It can be claimed freely from the AlphaSwap faucet.
- WETH: 566805685906486115028880111939537255682077596399966799517328603082702059381 6field
    - decimals: 6
    - WETH is a token created by AlphaSwap for testing purposes.

    - It can be claimed freely from the AlphaSwap faucet.
- ALS-USDT: 1141512325107611790857269140861172710185589976304892130253971148356104918741field
    - decimals: 6
    - ALS-USDT is a pair token for the liquidity pair ALS + USDT.

Since an almost infinite number of tokens can be created on AlphaSwap, it would not be easy to get all the tokens an account has. Therefore, it is recommended to allow users to import their own tokens to be displayed in addition to these common tokens.

Query APIs

Get Token Info

The `tokens` mapping stores all meta info (`TokenInfo`) of all tokens in AlphaSwap.

- `key: field`: the token id
- `val: TokenInfo`: the token meta info

```
struct TokenInfo {
    name: field,  // the field in bytes is the same as the string
bytes of the name
    symbol: field, // the field in bytes is the same as the string
bytes of the symbol
    decimals: u8,
    total_supply: u128,
    admin: address,
    mintable: bool,
    burnable: bool,
}
```

API: `${aleo_rpc}/program/${alphaswap_pid}/mapping/tokens/${token_id}`

Example:

```
# Get the token info of USDT
curl -s
${aleo_rpc}/program/alphaswap_v1.aleo/mapping/tokens/725661112884578732791
551467370687855499176489412483327103550882637097088086 5field

# Response:
{
  name: 1431520340field,
  symbol: 1431520340field,
  decimals: 6u8,
  total_supply: 100000000000000000000u128,
  admin: aleo1uldp2afc9gnfsxd0r2svaecax8quutny5j6ns2qa80yp5uhsac9q35h7h6,
  mintable: false,
```

```
  burnable: false
}
```

Get Public Token Balance

The `account` mapping stores the public balances of all tokens and all accounts in AlphaSwap.

- `key: field`: `key = bhp256_hash_to_field({token: field, user: address})`
- `val: u128`: the public balance of the token owned by the account

API: `${aleo_rpc}/program/${alphaswap_pid}/mapping/account/${balance_key}`

Example:

```
# Get the USDT balance of address
aleo1p9zgcw9a2j5m32p2pjngf22hz54zfdwj2j0zpy5d3c3ys70umy8sfrnuml
# Need to calculate the balance key first:
#   balance_key
#     = bhp256_hash_to_field("{token:
72566111288457873279155146737068785549917648941248332710355088263709708808
65field, user:
aleo1p9zgcw9a2j5m32p2pjngf22hz54zfdwj2j0zpy5d3c3ys70umy8sfrnuml}")
#     =
28606490758176343585657707751693703844368499097075673840301388901218134893
66field

curl -s
${aleo_rpc}/program/alphaswap_v1.aleo/mapping/account/28606490758176343585
65770775169370384436849909707567384030138890121813489366field

# Response:
10000000000u128
```

Get Approvals

The `approvals` mapping stores authorization data of all tokens and all accounts.

- `key: field`: `key = bhp256_hash_to_field({token: field, payer: address, spender: address})`
- `val: u128`: the amount of public tokens can be spent by spender

API: `${aleo_rpc}/program/${alphaswap_pid}/mapping/approvals/${approval_key}`

Example:

```
# Get the USDT approval amount from
aleo1p9zgcw9a2j5m32p2pjngf22hz54zfdwj2j0zpy5d3c3ys70umy8sfrnuml to
aleo1ashyu96tjwe63u0gtnnv8z5lhapdu4l5pjsl2kha7fv7hvz2eqxs5dz0rg
# Need to calculate the approval key first:
```

```
#   approval_key
#     = bhp256_hash_to_field("{token:
72566111288457873279155146737068785549917648941248332710355088263709708808
65field, payer:
aleo1p9zgcw9a2j5m32p2pjngf22hz54zfdwj2j0zpy5d3c3ys70umy8sfrnuml, spender:
aleo1ashyu96tjwe63u0gtnnv8z5lhapdu4l5pjsl2kha7fv7hvz2eqxs5dz0rg}")
 #     =
30674912363542564958777465909638825830665336820647766319805696641552019217
71field

curl -s
${aleo_rpc}/program/alphaswap_v1.aleo/mapping/approvals/306749123635425649
587774659096388258306653368206477663198056966415520192177 1field

# Response:
20000u128
```

Get Private Token Record

Similar to the credits.aleo and ARC20 programs, all private tokens in AlphaSwap are stored in records (named `PrivateToken`). The difference is that AlphaSwap token record has an additional token_id to indicate which token it corresponds to.

Here is the record struct:

```
record PrivateToken {
    // The token owner
    owner: address,
    // The token id
    token: field,
    // The token amount
    amount: u128,
}
```

Transitions

transfer_public

`transfer_public` is used to transfer public tokens.

- It is similar to the transfer of aleo/arc20/erc20, but with an additional `token_id` parameter.

Transition:

```
transfer_public(public token_id: field, public to: address, public amount:
u128)
```

Params:

- `token_id: field`: the token id to be transferred
- `to: address`: the receiver address
- `amount: u128`: amount of tokens to be transferred

transfer_public_to_private

`transfer_public_to_private` is used to transfer and convert public tokens to a new private token record (PrivateToken).

Transition:

```
transfer_public_to_private(public token_id: field, private to: address,
public amount: u128) -> PrivateToken
```

Params:

- `token_id: field`: the token id to be transferred
- `to: address`: the receiver address
- `amount: u128`: amount of tokens to be transferred and converted
- Output 1 `PrivateToken` record: a new `PrivateToken` record owned by `to`

transfer_private_to_public

`transfer_private_to_public` is used to transfer and convert a private token record (`PrivateToken`) to public tokens.

Transition:

```
transfer_private_to_public(private pt_in: PrivateToken, public to:
address, public amount: u128) -> PrivateToken
```

Params:

- `pt_in: PrivateToken`: the PrivateToken record to be spent
- `to: address`: the receiver address
- `amount: u128`: amount of tokens to be transferred and converted
- Output 1 `PrivateToken`: it is a new `PrivateToken` record (owned by the caller) with an amount of `pt_in.amount - amount`.

transfer_private

`transfer_private` is used to transfer private tokens (`PrivateToken` records).

Transition:

```
transfer_private(private pt_in: PrivateToken, private to: address, private
amount: u128) -> (PrivateToken, PrivateToken)
```

Params:

- `pt_in: PrivateToken`: the PrivateToken record to be spent
- `to: address`: the receiver address
- `amount: u128`: amount of tokens to be transferred
- Output 2 PrivateToken records: the first belongs to the receiver(`to`), the second is a change belonging to the caller

approve_public

`approve_public` is used to increase the amount of an approval.

Transition:

```
approve_public(public token_id: field, public spender: address, public
amount: u128)
```

Params:

- `token_id: field`: the token id to be approved
- `spender: address`: the spender address
- `amount: u128`: the maximum amount that the spender can spend

unapprove_public

`unapprove_public` is used to decrease the amount of an approval.

Transition:

```
unapprove_public(public token_id: field, public spender: address, public
amount: u128)
```

Params:

- `token_id: field`: the token id to be approved
- `spender: address`: the spender address
- `amount: u128`: the maximum amount that the spender can spend

transfer_from_public

`transfer_from_public` is used to transfer public tokens from other accounts.

- It is similar to the ERC20 transfer_from, but with an additional `token_id` parameter.

Transition:

```
transfer_from_public(public token_id: field, public from: address, public
to: address, public amount: u128)
```

Params:

- `token_id: field`: the token id to be transferred
- `from: address`: the address of the account from which the token is transferred
- `to: address`: the receiver address
- `amount: u128`: amount of tokens to be transferred

join

`join` is used to merge two `PrivateToken` records into a new `PrivateToken` record.

- The two records being joined must have the same owner and the token id.

Transition:

```
join(private pt1: PrivateToken, private pt2: PrivateToken) -> PrivateToken
```

Params:

- `pt1: PrivateToken`: the PrivateToken record to be spent
- `pt2: PrivateToken`: the PrivateToken record to be spent
- Output 1 PrivateToken record: the new record with an amount of `pt1.amount + pt2.amount`.