HEADS UP: Tutorials are now at chialisp.com

How to lock a coin with a custom puzzle

How to spend a coin using an RPC call

Create a puzzle

Create whatever puzzle you want for your coin. I'll use a password-locked coin as an example.

```
(mod (password new_puzhash amount)
   (defconstant CREATE_COIN 51)

   (if (= (sha256 password) (q . 0x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824))
      (list (list CREATE_COIN new_puzhash amount))
      (x)
   )
)
```

0x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 is sha256 of the word hello.

Example valid solution: (hello 0x351A85E8CBF8C6BC1619C6653B1930F975A806574FBEB6D06C61F33AB4BEA82B 2).
If password hello is correct, the puzzle will create a new coin with 2 mojos and lock it using a new puzzle hash 0x351A85E8CBF8C6BC1619C6653B1930F975A806574FBEB6D06C61F33AB4BEA82B. Any remaining change goes to a farmer as a fee.

Get puzzle hash from a puzzle

You can use Chialisp to get the hash of the Chialisp puzzle. The puzzle to get puzzle hash is following:

```
(mod (puzzle)
   (defconstant TREE 1)

   (defun sha256tree1 (TREE)
     (if (l TREE)
        (sha256 2 (sha256tree1 (f TREE)) (sha256tree1 (r TREE)))
        (sha256 1 TREE)
     )
   )

   (sha256tree1 puzzle)
)
```

A solution to this puzzle then needs to contain the compiled puzzle we want to hash in the first position (just wrap the compiled puzzle with parentheses and provide it as a solution). Compiled version of the puzzle can be obtained using https://clisp.surrealdev.com/.

Example solution for the password-locked coin:

```
((a (q 2 (i (= (sha256 5) (q . 0x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824)) (q 4 (c 2 (c 11 (c 23 ()))) ()) (q 8)) 1) (c (q . 51) 1)))
```

The resulting hash for this example puzzle is 0x4843c869bba5f65aa1e806cd372dae5668ca3b69640d067e86837ca96b324e71.

Convert puzzle hash to a receive address

We can use Chia explorer online tool for converting between puzzle hash to receive address and back. The puzzle hash is converted to bach32m format with `xch` prefix to form a receive address. Receive address for puzzle hash `0x4843c869bba5f65aa1e806cd372dae5668ca3b69640d067e86837ca96b324e71` is `xch1fppus6dm5hm94g0gqmxnwtdw2e5v5wmfvsxsvl5xsd72j6ejfecsdnkf2e`.

Send Chia to receive address

Use Chia GUI to send a transaction as you would typically do with the amount you want. As a receive address set address from the previous step (password-locked coin example: `xch1fppus6dm5hm94g0gqmxnwtdw2e5v5wmfvsxsvl5xsd72j6ejfecsdnkf2e`). That will lock your coin with a new puzzle.

Find your coin's parent info in Chia explorer

To find your newly created coin using Chia explorer, you can simply search for a wallet address. You'll find all coins that were locked using the same puzzle if you scroll down. Your coin should be the latest one.

Get serialized puzzle and solution

You can either use Quexington's Chialisp dev utility to calculate the puzzle's serialized value (just follow the README) or use an experimental online tool https://clisp.surrealdev.com/ which shows you serialized value when you compile your program.

To calculate the serialized value of the solution, you can use the same tools, but since the solution is not in a valid program, you first need to make it into one. That means wrapping it around with a quote. In the case of example solution `(hello 0x351A85E8CBF8C6BC1619C6653B1930F975A806574FBEB6D06C61F33AB4BEA82B 2)` you need to turn it into `(q . (hello 0x351A85E8CBF8C6BC1619C6653B1930F975A806574FBEB6D06C61F33AB4BEA82B 2))` and then you can serialize it.

Serialized version of password-locked coin puzzle is `ff02ffff01ff02ffff03ffff09ffff0bff0580ffff01a02cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b982480ffff01ff04ffff04ff02ffff04ff0bffff04ff17ff80808080ff8080ffff01ff088080ff0180ffff04ffff0133ff018080` and serialized version of its example solution is `ff8568656c6c6fffa0351a85e8cbf8c6bc1619c6653b1930f975a806574fbeb6d06c61f33ab4bea82bff0280`.

Spend coin with a custom puzzle

To spend your coin, you only need to call RPC ([broadcast transaction example]()).

```
curl --insecure --cert ~/.chia/mainnet/config/ssl/full_node/private_full_node.crt --key
~/.chia/mainnet/config/ssl/full_node/private_full_node.key -d '{        "spend_bundle": {
            "aggregated_signature":
"0xc0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
            "coin_solutions": [
                {
                    "coin": {
                        "amount": 1,
                        "parent_coin_info": "0xdeadbeef",
                        "puzzle_hash": "0x4843c869bba5f65aa1e806cd372dae5668ca3b69640d067e86837ca96b324e71"
                    },
                    "puzzle_reveal":
"ff02ffff01ff02ffff03ffff09ffff0bff0580ffff01a02cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b982480ffff01ff04fff
f04ff02ffff04ff0bffff04ff17ff80808080ff8080ffff01ff088080ff0180ffff04ffff0133ff018080 ",
                    "solution": "ff8568656c6c6fffa0351a85e8cbf8c6bc1619c6653b1930f975a806574fbeb6d06c61f33ab4bea82bff0280"
                }
            ]
        }}' -H "Content-Type: application/json" -X POST https://localhost:8555/push_tx
```

spend_bundle contains aggregated_signature, which we can later assert in the puzzle and coin_solutions for all coins we spend. If aggregated_signature is not necessary for your puzzle, use 0xc followed by 191 zeros (as in the example above). However, it's worth noting that a puzzle that doesn't use a signature is not safe and should be used only for testing purposes.

coin_solution contains information about the coin to locate it (amount, [parent_coin_info](), and puzzle_hash). It also contains a serialized puzzle as a puzzle_reveal and serialized solution.

If you fill in all your information correctly and send this request, your coin will be spent according to its provided solution.