

To make Medium work, we log user data.
By using Medium, you agree to our
Privacy Policy, including cookie policy.



Alvaro Andres Pinzon Cortes

Follow

May 5, 2022 · 2 min read · Listen



Save



Creating simple zero-knowledge verifier contract with ZoKrates (0.7.13) solidity (0.8.0)

Prerequisites

- Install [docker](#)

Step by step guide

1. Create a folder named **code** to put the code
2. Create the [square.zok file](#) with this content:

```
def main(private field a, field b):  
    assert(a * a == b)  
    return
```

3. Start docker

4. Open the terminal run this command:

```
docker run -v <PATH_TO_FOLDER>:/home/zokrates/code -ti  
zokrates/zokrates:0.7.13 /bin/bash
```

In my case that PATH_TO_FOLDER is where I created the [square.zok file](#) which is:

```
docker run -v /Users/andi/Downloads/UdacityGithubNeu-  
master/zokrates/code:/home/zokrates/code -ti
```

zokrates/zokrates:0.7.13 /bin/bash

To make Medium work, we log user data.

By using Medium, you agree to our

5. Change directory to `contracts` [Privacy Policy](#), including cookie policy.

`cd code`

6. Run this commands to create the proof and verifier

```
# compile
zokrates compile -i square.zok
# perform the setup phase
zokrates setup
# execute the program
zokrates compute-witness -a 337 113569
# generate a proof of computation
zokrates generate-proof
# export a solidity verifier
zokrates export-verifier
```

[Open in app](#) ↗

[Sign up](#)

[Sign In](#)



```
1 // This file is MIT Licensed.
2 //
3 // Copyright 2017 Christian Reitwiesner
4 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software
5 // The above copyright notice and this permission notice shall be included in all copies or substantial
6 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
7 pragma solidity ^0.8.0;
8 library Pairing {
9     struct G1Point {
10         uint X;
11         uint Y;
12     }
13     // Encoding of field elements is: X[0] * z + X[1]
14     struct G2Point {
15         uint[2] X;
16         uint[2] Y;
17     }
18     /// @return the generator of G1
19     function P1() pure internal returns (G1Point memory) {
20         return G1Point(1, 2);
21     }
22     /// @return the generator of G2
23     function P2() pure internal returns (G2Point memory) {
24         return G2Point(
25             [10857046999023057135944570762232829481370756359578518086990519993285655852781,
26              11559732032986387107991004021392285783925812861821192530917403151452391805634],
27             [8495653923123431417604973247489272438418190587263600148770280649306958101930,
28              4082367875863433681332203403145435568316851327593401208105741076214120093531]
29         );
30     }
31     /// @return the negation of p, i.e. p.addition(p.negate()) should be zero.
32     function negate(G1Point memory p) pure internal returns (G1Point memory) {
33         // The prime q in the base field F.q for G1
34         uint q = 21888242871839275222246405745257275088696311157297823662689037894645226208583
```

7. Compile

To make Medium work, we log user data.
By using Medium, you agree to our
Privacy Policy, including cookie policy.

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'SOLIDITY COMPILER' panel is active, displaying the compiler version '0.8.0+commit.c7dfd78e' and the language 'Solidity'. The 'EVM VERSION' is set to 'default'. The 'COMPILER CONFIGURATION' section includes options for 'Auto compile', 'Enable optimization' (set to 200), and 'Hide warnings'. A red box highlights the 'Compile verifier.sol' button. Below this, there are buttons for 'Compile and Run script', 'Publish on Ipfs', 'Publish on Swarm', and 'Compilation Details'. The main editor area shows the Solidity code for 'verifier.sol', which includes a library for pairing points on an elliptic curve. The code is as follows:

```
1 // This file is MIT Licensed.
2 //
3 // Copyright 2017 Christian Reitwiessner
4 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software
5 // The above copyright notice and this permission notice shall be included in all copies or
6 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
7 pragma solidity ^0.8.0;
8 library Pairing {
9     struct G1Point {
10         uint X;
11         uint Y;
12     }
13     // Encoding of field elements is: X[0] * z + X[1]
14     struct G2Point {
15         uint[2] X;
16         uint[2] Y;
17     }
18     /// @return the generator of G1
19     function P1() pure internal returns (G1Point memory) {
20         return G1Point(1, 2);
21     }
22     /// @return the generator of G2
23     function P2() pure internal returns (G2Point memory) {
24         return G2Point(
25             [1085704699902305713594457076223282948137075635957851808699051999328555852781,
26               11559732032986387107991004021392285783925812861821192530917403151452391805634],
27             [8495653923123431417604973247489272438418190587263600148770280649306958101930,
28               4082367875863433681332203403145435568316851327593401208105741076214120093531]
29         );
30     }
31     /// @return the negation of p, i.e. p.addition(p.negate()) should be zero.
32     function negate(G1Point memory p) pure internal returns (G1Point memory) {
33         // The prime q in the base field F_q for G1
34         uint q = 21888242871839275222246405745257275088696311157297823662689037894645226208583
```

8. Deploy

The screenshot shows the Remix Ethereum IDE interface, specifically the 'DEPLOY & RUN TRANSACTIONS' panel. The 'ENVIRONMENT' is set to 'JavaScript VM (Berlin)'. The 'ACCOUNT' is '0x5B3...eddC4 (99.999999%)'. The 'GAS LIMIT' is '3000000'. The 'VALUE' is '0 Wei'. The 'CONTRACT' is 'Verifier - contracts/verifier.sol'. A red box highlights the 'Deploy' button. Below this, there are options to 'Publish to IPFS' or 'At Address'. The 'Transactions recorded' section shows a list of transactions, including 'PAIRING AT 0X7EF...BCB47 (MEMORY)' and 'VERIFIER AT 0XDA0...42B53 (MEMORY)'. The main editor area shows the Solidity code for 'verifier.sol', which is the same as in the previous screenshot.

remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.0+commit.c7dfd78e.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM (Berlin)

ACCOUNT
0x5B3...eddC4 (99.999999%)

GAS LIMIT
3000000

VALUE
0 Wei

CONTRACT
Verifier - contracts/verifier.sol
Select contract for Deploy or At Address.
Deploy
Publish to IPFS

OR
At Address Load contract from Address

Transactions recorded 6

Deployed Contracts
PAIRING AT 0X7EF...BCB47 (MEMORY)
VERIFIER AT 0XDA0...42B53 (MEMORY)

To make Medium work, we log user data.
By using Medium, you agree to our Privacy Policy, including cookie policy.

```

1  pragma solidity ^0.8.0;
2  library Pairing {
3      struct G1Point {
4          uint X;
5          uint Y;
6      }
7      // Encoding of field elements is: X[0] * z + X[1]
8      struct G2Point {
9          uint[2] X;
10         uint[2] Y;
11     }
12     /// @return the generator of G1
13     function P1() pure internal returns (G1Point memory) {
14         return G1Point(1, 2);
15     }
16     /// @return the generator of G2
17     function P2() pure internal returns (G2Point memory) {
18         return G2Point(
19             [10857046999023057135944570762232829481370756359578518086990519993285655852781,
20               11559732032986387107991004021392285783925812861821192530917403151452391805634],
21             [8495653923123431417604973247489272438418190587263600148770280649306958101930,
22               4082367875863433681332203403145435568316851327593401208105741076214120093531]
23         );
24     }
25     /// @return the negation of p, i.e. p.addition(p.negate()) should be zero.
26     function negate(G1Point memory p) pure internal returns (G1Point memory) {
27         // The prime q in the base field F_q for G1
28         uint q = 2188824287183927522246405745257275088696311157297823662689037894645226208583
29     }
30 }
31
32 // This file is MIT Licensed.
33 // Copyright 2017 Christian Reitwiessner
34 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software
35 // The above copyright notice and this permission notice shall be included in all copies or
36 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
37
38 uint256[1] input = [
39     "113569"
40 ]

```

remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.0+commit.c7dfd78e.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM (Berlin)

ACCOUNT
0x5B3...eddC4 (99.999999%)

GAS LIMIT
3000000

VALUE
0 Wei

CONTRACT
Pairing - contracts/verifier.sol
Deploy
Publish to IPFS

OR
At Address Load contract from Address

Transactions recorded 6

Deployed Contracts
PAIRING AT 0X7EF...BCB47 (MEMORY)
VERIFIER AT 0XDA0...42B53 (MEMORY)

```

1  // This file is MIT Licensed.
2  // Copyright 2017 Christian Reitwiessner
3  // Permission is hereby granted, free of charge, to any person obtaining a copy of this software
4  // The above copyright notice and this permission notice shall be included in all copies or
5  // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
6  pragma solidity ^0.8.0;
7  library Pairing {
8      struct G1Point {
9          uint X;
10         uint Y;
11     }
12     // Encoding of field elements is: X[0] * z + X[1]
13     struct G2Point {
14         uint[2] X;
15         uint[2] Y;
16     }
17     /// @return the generator of G1
18     function P1() pure internal returns (G1Point memory) {
19         return G1Point(1, 2);
20     }
21     /// @return the generator of G2
22     function P2() pure internal returns (G2Point memory) {
23         return G2Point(
24             [10857046999023057135944570762232829481370756359578518086990519993285655852781,
25               11559732032986387107991004021392285783925812861821192530917403151452391805634],
26             [8495653923123431417604973247489272438418190587263600148770280649306958101930,
27               4082367875863433681332203403145435568316851327593401208105741076214120093531]
28         );
29     }
30     /// @return the negation of p, i.e. p.addition(p.negate()) should be zero.
31     function negate(G1Point memory p) pure internal returns (G1Point memory) {
32         // The prime q in the base field F_q for G1
33         uint q = 2188824287183927522246405745257275088696311157297823662689037894645226208583
34     }
35 }
36
37 // This file is MIT Licensed.
38 // Copyright 2017 Christian Reitwiessner
39 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software
40 // The above copyright notice and this permission notice shall be included in all copies or
41 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
42
43 uint256[1] input = [
44     "113569"
45 ]

```

9. The function verifyTx requires an array because structs in solidity map to arrays in the ABI:

To make Medium work, we log user data.
By using Medium, you agree to our
Privacy Policy, including cookie policy.

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

