

Create and verify aProof

Usegnark/backend

Once the [circuit](#) is [compiled](#) , you can run the three algorithms of a zk-SNARK back end:

- Setup
- Prove
- Verify

note Supported zk-SNARK backends are under `gnark/backend` . `gnark` currently implements `Groth16` and an experimental version of `PlonK` . Use a zk-SNARK back end * `Groth16` * `PlonK`

```
// 1. One time setup pk , vk , err := groth16 . Setup ( cs )
// 2. Proof creation proof , err := groth16 . Prove ( cs , pk , witness )
// 3. Proof verification err := groth16 . Verify ( proof , vk , publicWitness ) // 1. One time setup publicData ,
—
:= plonk . Setup ( cs ,
... )
// WIP
// 2. Proof creation proof , err := plonk . Prove ( r1cs , publicData , witness )
// 3. Proof verification err := plonk . Verify ( proof , publicData , publicWitness )
```

Construct the witness

Within a Go process, re-use the circuit data structure to construct the witness.

type Circuit struct

```
{ X frontend . Variable Y frontend . Variable gnark:" , public" }
```

assignment :=

& Circuit { X :

3 , Y :

35 , } witness ,

—

```
:= frontend . NewWitness ( assignment , ecc . BN254 ) // use the witness directly in zk-SNARK backend APIs groth16 .
Prove ( cs , pk , witness ) // test file --> assert.ProverSucceeded(cs, &witness) tip If witness is not built within the same
process, or in another programming language, refer to Serialize .
```

Verify aProof

on Ethereum

One `ecc.BN254 + Groth16` , `gnark` can export the `groth16.VerifyingKey` as a solidity smart contract.

Refer to [the code example](#) and [end-to-end integration test](#) using a `geth` simulated blockchain.

```
// 1. Compile (Groth16 + BN254) cs , err := frontend . Compile ( ecc . BN254 , r1cs . NewBuilder ,
& myCircuit )
// 2. Setup pk , vk , err := groth16 . Setup ( cs )
// 3. Write solidity smart contract into a file err = vk . ExportSolidity ( f Edit this page Last updated on Mar 2, 2023
```

