Groth16

A zero-knowledge proof allows a prover to validate that a statement is true without revealing any information about the inputs. For example, a prover can validate that they know the solution to a sudoku puzzle without revealing the solution.

Zero-knowledge succinct non-interactive argument of knowledge (zk-SNARKs) are a family of zero-knowledge proofs that are non-interactive, have succinct proof size and efficient verification time. An important and widely used variant of them is pairing-based zk-SNARKs like the Groth16 proof system, which is one of the most efficient and widely used.

The Move API in Sui enables you to verify any statement that can be expressed in a NP-complete language efficiently using Groth16 zk-SNARKs over either the BN254 or BLS12-381 elliptic curve constructions.

There are high-level languages for expressing these statements, such as Circom , used in the following example.

Groth16 requires a trusted setup for each circuit to generate the verification key. The API is not pinning any particular verification key and each user can generate their own parameters or use an existing verification to their apps.

The following example demonstrates how to create a Groth16 proof from a statement written in Circom and then verify it using the Sui Move API. The API currently supports up to eight public inputs.

In this example, we create a proof which demonstrates that we know a factorisation a * b = c of a publicly known number c without revealing a and b .

Assuming that the circom compiler has been installed , the above circuit is compiled using the following command:

This outputs the constraints in R1CS format and the circuit in Wasm format.

To generate a proof verifiable in Sui, you need to generate a witness. This example uses Arkworks' ark-circom Rust library. The code constructs a witness for the circuit and generates a proof for it for a given input. Finally, it verifies that the proof is correct.

Recall that this creates a proof that the prover knows a factorisation, in this case of the 5th Fermat number (2 32 + 1 = 4294967297 = 641 * 6700417).

The output of the above function will be

All these outputs are needed to verify the proof.

The API in Sui for verifying a proof expects a special processed verification key, where only a subset of the values are used. Ideally, computation for this prepared verification key happens only once per circuit. You can perform this processing using the sui::groth16::prepare_verifying_key method of the Sui Move API with a serialization of the params.vk value used previously.

The output of the prepare_verifying_key function is a vector with four byte arrays, which corresponds to the vk_gamma_abc_g1_bytes , alpha_g1_beta_g2_bytes , gamma_g2_neg_pc_bytes , delta_g2_neg_pc_bytes .

To verify a proof, you also need two more inputs, public_inputs_bytes and proof_points_bytes , which are printed by the program above.

The following example smart contract uses the output from the program above. It first prepares a verification key and verifies the corresponding proof. This example uses the BN254 elliptic curve construction, which is given as the first parameter to the prepare_verifying_key and verify_groth16_proof functions. You can use the bls12381 function instead for BLS12-381 construction.

Usage

The following example demonstrates how to create a Groth16 proof from a statement written in Circom and then verify it using the Sui Move API. The API currently supports up to eight public inputs.

In this example, we create a proof which demonstrates that we know a factorisation a * b = c of a publicly known number c without revealing a and b .

Assuming that the circom compiler has been installed , the above circuit is compiled using the following command:

This outputs the constraints in R1CS format and the circuit in Wasm format.

To generate a proof verifiable in Sui, you need to generate a witness. This example uses Arkworks' ark-circom Rust library. The

code constructs a witness for the circuit and generates a proof for it for a given input. Finally, it verifies that the proof is correct.

Recall that this creates a proof that the prover knows a factorisation, in this case of the 5th Fermat number (2 32 + 1 = 4294967297 = 641 * 6700417).

The output of the above function will be

All these outputs are needed to verify the proof.

The API in Sui for verifying a proof expects a special processed verification key, where only a subset of the values are used. Ideally, computation for this prepared verification key happens only once per circuit. You can perform this processing using the sui::groth16::prepare_verifying_key method of the Sui Move API with a serialization of the params.vk value used previously.

The output of the prepare_verifying_key function is a vector with four byte arrays, which corresponds to the vk_gamma_abc_g1_bytes , alpha_g1_beta_g2_bytes , gamma_g2_neg_pc_bytes , delta_g2_neg_pc_bytes .

To verify a proof, you also need two more inputs, public_inputs_bytes and proof_points_bytes , which are printed by the program above.

The following example smart contract uses the output from the program above. It first prepares a verification key and verifies the corresponding proof. This example uses the BN254 elliptic curve construction, which is given as the first parameter to the prepare_verifying_key and verify_groth16_proof functions. You can use the bls12381 function instead for BLS12-381 construction.