

Serialize

gnark

objects

gnark objects implementio.WriterTo andio.ReaderFrom interfaces.

To serialize agnark object:

```
// compile a circuit cs , err := frontend . Compile ( ecc . BN254 , r1cs . NewBuilder ,
```

& circuit)

// cs implements io.WriterTo var buf bytes . Buffer cs . WriteTo (& buf) To deserialize, first instantiate a curve-typed object, as pergnark API design choices, these objects are not directly accessible (underinternal/).

```
// instantiate a curve-typed object cs := groth16.NewCS ( ecc . BN254 ) // cs implements io.ReaderFrom cs . ReadFrom ( & buf ) caution Constraint systems (R1CS and SparseR1CS , for Groth16 and PLONK ) currently use thecbor serialization protocol.
```

Other gnark objects, like `ProvingKey`, `VerifyingKey` or `Proof` contains elliptic curve points, and use a binary serialization protocol, allowing [point compression](#).

We strongly discourage to using another protocol, because for security reasons, deserialization must also perform curve and subgroup checks.

Compression

Elliptic curve points, which are the main citizens of `ProvingKey`, `VerifyingKey` or `Proof` objects can be compressed by storing only the `x` coordinate, and a parity bit. This divides the required bytes that represent these objects by two, but comes at a significant CPU cost on the deserialization side.

These objects implement `io.WriterRawTo`, which doesn't use point compression.

```
provingKey . WriteRawTo ( & buf )
```

```
// alternatively, provingKey.WriteTo(&buf) ... pk := groth16 . NewProvingKey ( ecc . BN254 ) pk . ReadFrom ( & buf )
```

// reader will detect if points are compressed or not. tip UseWriteRawTo when deserialization speed >>> storage cost, otherwise useWriteTo with point compression.

Witness

Witnesses (inputs to the `Prove` or `Verify` functions) may be constructed outside of `gnark`, in a non-Go codebase.

Two types of witnesses exist:

- Full witness
 - : contains public and secret inputs, needed by Prove
- Public witness
 - : contains public inputs only, needed by Verify

For performance reason (witnesses can be large), witnesses should be encoded using a binary protocol. For convenience, gnark also support JSON encoding.

Binary protocol

While there is no standard yet, we followed similar patterns used by other zk-SNARK libraries.

```
// Full witness -> [uint32(nbElements) | publicVariables | secretVariables] // Public witness -> [uint32(nbElements) | publicVariables ] Where:
```

- `nbElements == len(publicVariables) + len(secretVariables)`
- `.`
- each variable (a field element
- `)` is encoded as a big-endian byte array, where `len(bytes(variable)) == len(bytes(modulus))`

Ordering

The ordering sequence is first, `publicVariables` , then `secretVariables` . Each subset is ordered from the order of definition in the circuit structure.

For example, with this circuit `onecc.BN254` :

```
type Circuit struct
```

{ X frontend . Variable Y frontend . Variable_{gnark:"public"} Z frontend . Variable } A valid witness would be:

- [illegible]

Example

This example is intended for a multi-process usage of `gnark` where you need to construct the witness in one process and deserialize it in another.

tip If the witness creation and proof creation live in the same process, refer to [Construct the witness](#). Full witness in Go // witness var assignment cubic . Circuit assignment . X =

3 assignment , Y =

35 witness ,

```
:= frontend . NewWitness ( & assignment , ecc . BN254 )
```

```
// Binary marshalling data , err := witness . MarshalBinary ( )
```

```
// JSON marshalling json , err := witness . MarshalJSON ( )
```

```
... // recreate a witness witness , err := witness . New ( ecc . BN254 , ccs . GetSchema ( ) )
```

```
// note that schema is optional for binary encoding
```

```
// Binary unmarshalling err := witness . UnmarshalBinary ( data )
```

```
// JSON unmarshalling err := witness . UnmarshalJSON ( json )
```

```
// extract the public part only publicWitness .
```

[= witness](#) . [Public \(\)](#) [Edit this page](#) Last updated on Mar 2, 2023 by aybehrouz [Previous](#) [Create and verify proofs](#) [Next](#) [Profile circuits](#) * [gnark objects](#) * [Compression](#) * [Witness](#) * * [Binary protocol](#) * * [Ordering](#) * * [Example](#)