

# Formalising Groth16 in Lean 4

Daniel Rogozin, for Yatima Inc

August 16, 2022\*

## 1 Introduction

In this document, we describe the Groth16 soundness formalisation in Lean 4. The text contains the protocol description as well as some comments to its implementation.

Groth16 is a kind of ZK-SNARK protocol introduced in [1]. The latter means that:

- It is *zero-knowledge*. In other words, a prover only has a particular piece of information.
- It is *non-interactive* in order to make secret parameters reusable.

Protocols of this kind have the core characteristics such as:

- *Soundness*, i.e., if a statement does not hold, then the prover cannot convince the verifier.
- *Completeness*, i.e., the verifier is convinced whenever a statement is true.
- *Zero-knowledge*, i.e., the only thing is needed is the truth of a statement.

Generally, non-interactive zero-knowledge proofs relies on the *common reference string* model, that is, a model where a public string is generated in a trusted way and all parties have an access to it.

Let us describe the common scheme that non-interactive zero-knowledge protocols obey, see [2] and [3] to have more details. Before that, we need a bit of terminology.

Let  $p \in F[X]$  be a polynomial, a prover is going to convince a verifier that they know  $p$ . In turn, knowing  $p$  means that a prover knows some of its roots. As it is well-known, any polynomial might be decomposed as follows whenever it has roots (since fields we consider are finite and they are not algebraically closed):

$$p(x) = \prod_{i < \deg(p)} (x - a_i) \tag{1}$$

for some  $a_i$ ,  $i < \deg(p)$ .

Assume that a prover has some values  $\{r_i \mid i < n\}$  where each  $r_i \in F$  for some  $n \leq \deg(p)$ . A prover wants to convince a verifier that  $p(r_i)$  for each  $a_i$  from that set.

If there  $a_i$ 's are really roots of  $p$ , then the polynomial  $p$  can be rewritten as:

$$p(x) = \left( \prod_{i < n} (x - r_i) \right) \cdot h(x) \tag{2}$$

for some  $h \in F[X]$ .

Denote  $(\prod_{i < n} (x - r_i))$  as  $t(x)$ . We shall call  $t(x)$  further a *target polynomial*. So, a verifier accepts only if a target polynomial  $t$  divides  $p$ , in particular, that means that all those  $r_i$ 's are roots of  $p$ .

---

\*This document may be updated frequently.

The next notion we need is a square span program (see [4]) for verification of which the target polynomial is used. Originally, it has been introduced as a simpler version of quadratic span programs for an alternative characterisation of NP.

- 
- 
- 
- 

Now we discuss specific aspects of Groth16 in addition the aforescribed general ZK-SNARK scheme. We emphasise such properties of Groth16 as:

- 
- 
- 
- 

## 2 Preliminary definitions

We have a fixed finite field  $F$ , and  $F[X]$  stands for the ring of polynomials over  $F$  as usual. The corresponding listing written in Lean:

```
variable {F : Type u} [field : Field F]
```

TODO: listings are rather ugly

In Groth16, we have random values  $\alpha, \beta, \gamma, \delta \in F$  that we introduce separately as a data type:

```
inductive Vars : Type
| alpha : Vars
| beta : Vars
| gamma : Vars
| delta : Vars
```

We also introduce the following parameters:

- $n_{stmt} \in \mathbb{N}$  — the statement size;
- $n_{wit} \in \mathbb{N}$  — the witness size;
- $n_{var} \in \mathbb{N}$  — the number of variables.

In Lean 4, we introduce those parameters as variables in the following way:

```
variable {n_stmt n_wit n_var : Nat}
```

We also define several finite collections of polynomials from the square span program:

- $u_{stmt} = \{f_i \in F[X] \mid i < n_{stmt}\}$
- $u_{wit} = \{f_i \in F[X] \mid i < n_{wit}\}$
- $v_{stmt} = \{f_i \in F[X] \mid i < n_{stmt}\}$
- $v_{wit} = \{f_i \in F[X] \mid i < n_{wit}\}$
- $w_{stmt} = \{f_i \in F[X] \mid i < n_{stmt}\}$
- $w_{wit} = \{f_i \in F[X] \mid i < n_{wit}\}$

We introduce those collections in Lean 4 as variables as well:

```
variable {u_stmt : Fin n_stmt → F[X]}
variable {u_wit : Fin n_wit → F[X]}
variable {v_stmt : Fin n_stmt → F[X]}
variable {v_wit : Fin n_wit → F[X]}
variable {w_stmt : Fin n_stmt → F[X]}
variable {w_wit : Fin n_wit → F[X]}
```

Let  $(r_i)_{i < n_{wit}}$  be a collection of elements of  $F$  (that is, each  $r_i \in F$ ) parametrised by elements of  $\{0, \dots, n_{wit}\}$ . Define the target polynomial  $t \in F[X]$  of degree  $n_{wit}$  as:

$$t = \prod_{i < n_{wit}} (x - r_i).$$

Clearly, these  $r_i$ 's are roots of  $t$ . The definition in Lean 4:

```
variable (r : Fin n_wit → F)

def t : F[X] := \prod i in finRange n_wit ,
  (x : F[X]) - Polynomial.c (r i)
```

We think of the collection  $\mathbf{r}$  as roots of the polynomial  $\mathbf{t}$  as it can be observed from the definition.

The polynomial  $t$  has the following self-evident properties:

**Lemma 1.**

1.  $\deg(t) = n_{wit}$ ;
2.  $t$  is monic, that is, its leading coefficient is equal to 1;
3. If  $n_{wit} > 0$ , then  $\deg(t) > 0$ .

We formalise these statements as follows (but we skip proofs):

```
lemma nat_degree_t : (t r).natDegree = n_wit
lemma monic_t : Polynomial.Monic (t r)
lemma degree_t_pos (hm : 0 < n_wit) : 0 < (t r).degree
```

Let  $\{a_{wit_i} \mid i < n_{wit}\}$  and  $\{a_{stmt_i} \mid i < n_{stmt}\}$  be collections of elements of  $F$ . A statement witness polynomial pair is a pair of single variable polynomials  $(F_{wit_{sv}}, F_{stmt_{sv}})$  such that  $F_{wit_{sv}}, F_{stmt_{sv}} \in F[X]$  and

- $F_{wit_{sv}} = \sum_{i < n_{wit}} a_{wit_i} u_{wit_i}(x)$
- $F_{stmt_{sv}} = \sum_{i < n_{stmt}} a_{stmt_i} u_{stmt_i}(x)$

Their Lean 4 counterparts:

```
def V_wit_sv (a_wit : Fin n_wit → F) : Polynomial F :=
  \sum i in finRange n_wit , a_wit i \bullet u_wit i

def V_stmt_sv (a_stmt : Fin n_stmt → F) : Polynomial F :=
  \sum i in finRange n_stmt , a_stmt i \bullet u_stmt i
```

Define the polynomial  $sat$  as:

$$\begin{aligned}
sat = (V_{stmt_{sv}} + V_{wit_{sv}}) \cdot & \\
& \cdot ((\sum_{i < n_{stmt}} a_{stmt_i} v_{stmt_i}(x)) + (\sum_{i < n_{wit}} a_{wit_i} v_{wit_i}(x))) - \\
& - ((\sum_{i < n_{stmt}} a_{stmt_i} w_{stmt_i}(x)) + (\sum_{i < n_{wit}} a_{wit_i} w_{wit_i}(x))) \quad (3)
\end{aligned}$$

A pair  $(F_{wit_{sv}}, F_{stmt_{sv}})$  satisfies *the square span program*, if the remainder of division of  $sat$  by  $t$  is equal to 0. This requirement is common for ZK-SNARK protocols as we discussed in the first section.

The Lean 4 analogue of the property defined above:

```

def satisfying (a_stmt : Fin n_stmt → F) (a_wit : Fin n_wit → F) :=
  (((\sum i in finRange n_stmt, a_stmt i \bullet u_stmt i)
    + \sum i in finRange n_wit, a_wit i \bullet u_wit i)
  *
  ((\sum i in finRange n_stmt, a_stmt i \bullet v_stmt i)
    + \sum i in finRange n_wit, a_wit i \bullet v_wit i)
  -
  ((\sum i in finRange n_stmt, a_stmt i \bullet w_stmt i)
    + \sum i in finRange n_wit, a_wit i \bullet w_wit i) : F[X]) %m (t r) = 0

```

### 3 Common reference string elements

Assume we interpreted  $\alpha, \beta, \gamma$ , and  $\delta$  somehow with elements of  $F$ , say  $crs_\alpha, crs_\beta, crs_\gamma$ , and  $crs_\delta$ , that is, in Lean 4:

```

def crs_alpha (f : Vars → F) : F := f Vars.alpha
def crs_beta (f : Vars → F) : F := f Vars.beta
def crs_gamma (f : Vars → F) : F := f Vars.gamma
def crs_delta (f : Vars → F) : F := f Vars.delta

```

For simplicity, we write this interpretation as a function  $f : \{\alpha, \beta, \gamma, \delta\} \rightarrow F$  defined by equations:

$$f(a) = crs_a \text{ for } a \in \{\alpha, \beta, \gamma, \delta\}.$$

In addition to those four elements of  $F$  we have a collection of degrees for  $a \in F$ :

$$\{a^i \mid i < n_{var}\}$$

formalised as:

```

def crs_powers_of_x (i : Fin n_var) (a : F) : F := (a)^(i : Nat)

```

We also introduce collections  $crs_l, crs_m$ , and  $crs_n$  for  $a \in F$ :

$$crs_l = \frac{((f(\beta)/f(\gamma)) \cdot (u_{stmt_i})(a)) + ((f(\alpha)/f(\gamma)) \cdot (v_{stmt_i})(a)) + w_{stmt_i}(a)}{f(\gamma)} \quad \text{for } i < n_{stmt} \quad (4)$$

$$crs_l = \frac{((f(\beta)/f(\delta)) \cdot (u_{wit_i})(a)) + ((f(\alpha)/f(\delta)) \cdot (v_{wit_i})(a)) + w_{wit_i}(a)}{f(\delta)} \quad \text{for } i < n_{wit} \quad (5)$$

$$crs_l = \frac{a^i \cdot t(a)}{f(\delta)}, \text{ for } i < n_{var} \quad (6)$$

Their Lean 4 version:

```
def crs_l (i : Fin n_stmt) (f : Vars → F) (a : F) : F :=
  ((f Vars.beta / f Vars.gamma) * (u_stmt i).eval (a)
  +
  (f Vars.alpha / f Vars.gamma) * (v_stmt i).eval (a)
  +
  (w_stmt i).eval (a)) / f Vars.gamma.

def crs_m (i : Fin n_wit) (f : Vars → F) (a : F) : F :=
  ((f Vars.beta / f Vars.delta) * (u_wit i).eval (a)
  +
  (f Vars.alpha / f Vars.delta) * (v_wit i).eval (a)
  +
  (w_wit i).eval (a)) / f Vars.delta

def crs_n (i : Fin (n_var - 1)) (f : Vars → F) (a : F) : F :=
  ((a)^(i : Nat)) * (t r).eval a / f Vars.delta
```

Assume we have fixed elements of a field  $A_\alpha, A_\beta, A_\gamma, A_\delta, B_\alpha, B_\beta, B_\gamma, B_\delta, C_\alpha, C_\beta, C_\gamma, C_\delta$ .

We also have indexed collections  $\{A_x \in F \mid x < n_{var}\}, \{B_x \in F \mid x < n_{var}\}, \{C_x \in F \mid x < n_{var}\}, \{A_l \in F \mid l < n_{stmt}\}, \{B_l \in F \mid l < n_{stmt}\}, \{C_l \in F \mid l < n_{stmt}\}, \{A_m \in F \mid m < n_{wit}\}, \{B_m \in F \mid m < n_{wit}\}, \{C_m \in F \mid m < n_{wit}\}, \{A_h \in F \mid h < n_{var-1}\}, \{B_h \in F \mid h < n_{var-1}\}, \{C_h \in F \mid h < n_{var-1}\}.$

```
variable { A_alpha A_beta A_gamma A_delta : F }
variable { B_alpha B_beta B_gamma B_delta : F }
variable { C_alpha C_beta C_gamma C_delta : F }
variable { A_x B_x C_x : Fin n_var → F }
variable { A_l B_l C_l : Fin n_stmt → F }
variable { A_m B_m C_m : Fin n_wit → F }
variable { A_h B_h C_h : Fin (n_var - 1) → F }
```

The adversary's proof representation is defined as the following three sums, for  $x \in F$ :

$$\begin{aligned} A = & A_\alpha \cdot crs_\alpha + A_\beta \cdot crs_\beta + A_\gamma \cdot crs_\gamma + A_\delta \cdot crs_\delta + \\ & + \sum_{i < n_{var}} A_{x_i} * x^i + \sum_{i < n_{stmt}} A_{l_i} * crs_l(x) + \\ & + \sum_{i < n_{wit}} A_{m_i} * crs_m(x) + \sum_{i < n_{var-1}} A_{h_i} * crs_n(x) \quad (7) \end{aligned}$$

$$\begin{aligned}
B = & B_\alpha \cdot crs_\alpha + B_\beta \cdot crs_\beta + B_\gamma \cdot crs_\gamma + B_\delta \cdot crs_\delta + \\
& + \sum_{i < n_{var}} B_{x_i} * x^i + \sum_{i < n_{stmt}} B_{l_i} * crs_{l_i}(x) + \\
& + \sum_{i < n_{wit}} B_{m_i} * crs_{m_i}(x) + \sum_{i < n_{var}-1} B_{h_i} * crs_n(x) \quad (8)
\end{aligned}$$

$$\begin{aligned}
C = & C_\alpha \cdot crs_\alpha + C_\beta \cdot crs_\beta + C_\gamma \cdot crs_\gamma + C_\delta \cdot crs_\delta + \\
& + \sum_{i < n_{var}} C_{x_i} * x^i + \sum_{i < n_{stmt}} C_{l_i} * crs_{l_i}(x) + \\
& + \sum_{i < n_{wit}} C_{m_i} * crs_{m_i}(x) + \sum_{i < n_{var}-1} C_{h_i} * crs_n(x) \quad (9)
\end{aligned}$$

Here, we provide the Lean 4 version of  $A$  only.

```

def A (f : Vars → F) (x : F) : F :=
  (A_alpha * crs_alpha F f)
  +
  A_beta * crs_beta F f
  +
  A_gamma * crs_gamma F f
  +
  A_delta * crs_delta F f
  +
  \sum i in (finRange n_var), (A_x i) * (crs_powers_of_x F i x)
  +
  \sum i in (finRange n_stmt),
    (A_l i) * (@crs_l F field n_stmt u_stmt v_stmt w_stmt i f x)
  +
  \sum i in (finRange n_wit),
    (A_m i) * (@crs_m F field n_wit u_wit v_wit w_wit i f x)
  +
  \sum i in (finRange (n_var - 1)), (A_h i) * (crs_n F r i f x)

```

A proof is called *verified*, if the following equation holds:

$$A \cdot B = crs_\alpha \cdot crs_\beta + \left( \sum_{i < n_{stmt}} a_{stmt_i} \cdot crs_{l_i}(x) \right) \cdot crs_\gamma + C \cdot crs_\delta \quad (10)$$

```

def verified (f : Vars → F) (x : F) (a_stmt : Fin n_stmt → F) : Prop :=
  A f x * B f x =
    (crs_alpha F f * crs_beta F f) +
    ((\sum i in finRange n_stmt, (a_stmt i) * @crs_l i f x) *
     (crs_gamma F f) + C f x * (crs_delta F f))

```

## 4 Modified common reference string elements

We modify common reference string elements from the previous section as multivariate polynomials.

## 5 Coefficient lemmas

## 6 Formalised soundness

## References

- [1] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.
- [2] Maksym Petkus. Why and how zk-snark works: Definitive explanation. *arXiv preprint arXiv:1906.07221*, 2019.
- [3] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.
- [4] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct nizk arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 532–550. Springer, 2014.