Groth16 notes

Karthik Inbasekar[a]

[a] *Ingonyama*

E-mail: karthik@ingonyama.com

ABSTRACT: A note on the Groth16 [1] protocol and some potential optimizations

1 Groth 16 Vanilla Protocol

1.1 Arithmetization

Consider an arithmetic circuit $C(X)$, that consists of addition and multiplication gates operating over a finite field $\mathbb{F}$. Consider sets of vectors u_q, v_q, w_q that encode the selection of the gates, and let the wires that connect the gates be $\{a_0, a_1 \ldots, a_m\}$, such that $a_0 = 1$. Here $\{q = 0, 1, \ldots n-1\}$ refer to the number of constraints in the circuit. Constraints that relate such gates and the wiring of the circuit are typically expressible in the form of $R1CS$ vector equations

$$< u_q, a > \circ < v_q, a >=< w_q, a > \tag{1.1}$$

where $< u, a >$ refers to the scalar product of two vectors and there are n such constraints. A simple multiplication gate constraint $a_i \cdot a_j = a_k$ is expressed using the above equation with the vectors with the assignments

$$\begin{aligned} u_q &= (0, 0, \ldots, (u_q)_i = 1, \ldots, 0) \\ v_q &= (0, 0, \ldots, (v_q)_j = 1, \ldots, 0) \\ w_q &= (0, 0, \ldots, (w_q)_k = 1, \ldots, 0) \end{aligned} \tag{1.2}$$

Substituting these vectors into (1.1) we see that the multiplication constraint $a_i \cdot a_j = a_k$ is reproduced. Since the set of constraints (1.1) is a linear system, obviously one can combine them into a set of matrices A, B, C of dimensions $n \times m$ such that

$$(A.a) \circ (B.a) = (C.a) \tag{1.3}$$

the a are input wire vectors consisting of public and witness elements.

We will use the following notation to label the wire vectors (with $a_0 = 1$)

$$\{a_0 | \underbrace{a_1, a_2, \ldots a_l}_{\text{Public } \in \mathbb{F}^l} | \underbrace{a_{l+1}, \ldots a_m}_{\text{Witness } \in \mathbb{F}^{m-l}} \} \tag{1.4}$$

If the maximum number of constraints are n (essentially the number of equations in (1.1)) then we pick a multiplicative subgroup H of $\mathbb{F}$ with domain size n. The target polynomial or the vanishing polynomial is the maximally reducible polynomial in H, i.e

$$t(X) = \prod_{i=0}^{n-1} (X - \omega^i) \equiv X^n - 1 \tag{1.5}$$

where $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$ span H and ω are primitive n th roots of unity.

Below we describe the method to encode, the circuit into a polynomial form. This is called the QAP (Quadratic arithmetic program) (see fig 1) It consists of polynomial

$$\begin{array}{cc} \xrightarrow{\qquad\qquad m \qquad\qquad} & \\ U = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 1 & 1 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & 1 & \dots & 1 \end{pmatrix} & \Big\downarrow n \end{array}$$

$$U_i(\omega^q)$$

Figure 1. Polynomial representation of the circuit: QAP

interpolations of the vectors u_q, v_q, w_q such that

$$A_i(\omega^q) = (u_q)_i \ , \ B_i(\omega^q) = (v_q)_i \ , \ C_i(\omega^q) = (w_q)_i \ \forall \ i = 0, 1, \ldots m \ , \ q = 0, 1, 2, \ldots n-1 \tag{1.6}$$

where $A_i(X), B_i(X), C_i(X)$ are the Lagrange interpolations respectively. Using the above, (1.1) is represented as

$$\begin{aligned} \sum_{i=0}^{m} a_i A_i(X) \cdot \sum_{i=0}^{m} a_i B_i(X) &\equiv \sum_{i=0}^{m} a_i C_i(X) \mod t(X) \\ \sum_{i=0}^{m} a_i A_i(X) \cdot \sum_{i=0}^{m} a_i B_i(X) &\equiv \sum_{i=0}^{m} a_i C_i(X) + h(X) \cdot t(X) \end{aligned} \tag{1.7}$$

Note that by construction

$$deg(t(X)) = n \ , \ deg(A_i(X)) = n-1 \ , \ deg(B_i(X)) = n-1 \ , \ deg(C_i(X)) = n-1 \tag{1.8}$$

where $h(X)$ is the quotient polynomial and since $deg(t(X)) = n$, then it follows from (1.7) that $deg(h(X)) = n-2$. Thus a Quadratic Arithmetic Program generates the polynomial representation for a given circuit.

1.2 Setup phase

In the setup phase, the goal is to generate a public CRS (Common Reference String) via an alaborate setup ceremony [2] usually via a Multi Party Computation scheme (MPC).MPC's are interactive protocols that involve several parties that contribute randomness in a serial fashion. In the end, the original random scalars with each of the participant referred to as "toxic waste" has to be destroyed for the scheme to function for Zero Knowledge. For example, several parties can participate in multiplication of a scalar (or its powers) with a Elliptic curve group element g, and as long as the secret scalar is not revealed or even if there is just one honest participant, the discrete log property guarantees that the process is non-invertible.

In order to implement this often one needs to choose Elliptic curves that are pairing friendly over some finite field $\mathbb{F}_{q^k}$. In particular q is prime and the curve supports groups $\mathbb{G}_1, \mathbb{G}_2$ each of order $|G_1| = |G_2| = r$ and the pairing equation $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ representing a bilinear map. The group $G_T \in F^*_{q^{k'}}$ is referred to as a target group (multiplicative group) and also has the property $|G_T| = r$ [1]. Thus, the pairing equation maps a product of two elliptic curve group elements in G_1 and G_2 respectively to a root of unity in an large extension field.

For example in the curve $BLS12-381$ [3]

$$\begin{aligned}
&(x, y) \in G_1 \subset E(F_q) \text{ with the equation} : y^2 = x^3 + 4\\
&(x, y) \in G_2 \subset E\left(F_{q^2}\right) \text{ with the equation} : y^2 = x^3 + 4(1 + i)\\
&G_T \equiv F^*_{q^{12}} \subset F_{q^{12}}
\end{aligned} \tag{1.9}$$

The field extensions in the BLS12 case are constructed/deconstructed recursively as

$$\begin{aligned}
F_{q^2} &\equiv \frac{F_q[X]}{X^2+1} \ ; \ eg: \ a_0 + a_1 X \text{ where } a_i \in F_q\\
F_{q^6} &\equiv \frac{F_{q^2}[\tilde{X}]}{\tilde{X}^3 - X - 1} \ ; \ eg: \ b_0 + b_1 \tilde{X} + b_2 \tilde{X}^2 \text{ where } b_i \in F_{q^2}\\
F_{q^{12}} &\equiv \frac{F_{q^6}[\hat{X}]}{\hat{X}^2 - \tilde{X}} \ ; \ eg: \ c_0 + c_1 \hat{X} \text{ where } c_i \in F_{q^6}
\end{aligned} \tag{1.10}$$

Denoting the generators of the groups $\mathbb{G}_1, \mathbb{G}_2$ respectively as g_1, g_2, we write the generator of the bilinear product as $g_T = e(g_1, g_2)$, which represents a multiplication rule such that

$$e(g_1, g_2) \neq 1 \ \text{ for } g_1 \neq 1 \ , \ g_2 \neq 1 \tag{1.11}$$

We normally represent multiplication with group elements using the notation

$$[a]_1 = a \cdot g_1 \ , \ [b]_2 = b \cdot g_2 \ , \ [c]_T = c \cdot g_T = c \cdot e(g_1, g_2) \tag{1.12}$$

Note that these are elliptic curve additions, i.e $[a]_1 = a.g_1 = g_1 + g_1 + \ldots$ a times. The additive homomorphisms ensures linearity, i.e one can verify that $[a+b]_1 = [a]_1 + [b]_1$ holds. Multiplications $c = a.b$ can be verified using the pairing equation $e([a]_1, [b]_2) = e([c]_1, [1]_2)$ where $[1]_2 = 1 \cdot g_2 = g_2$. All these properties generalize to vector multiplications as well.

The setup phase has the following steps

[1]Here we assume type III pairings, where there is no efficient isomorphism that maps an element in G_1 to element in G_2

1. define $\tau = (\alpha, \beta, \gamma, \delta, x) \in \mathbb{F}_r$ (scalar field)

2. Using τ compute

$$
\begin{aligned}
\sigma_1 &= \left(\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, vk', pk', \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2}\right) \\
\sigma_2 &= \left(\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1}\right) \\
pk' &= \left\{\frac{\beta A_i(x) + \alpha B_i(x) + C_i(x)}{\delta}\right\}_{i=l+1}^{m} \\
vk' &= \left\{\frac{\beta A_i(x) + \alpha B_i(x) + C_i(x)}{\gamma}\right\}_{i=0}^{l}
\end{aligned}
\tag{1.13}
$$

3. Then compute the CRS

$$
\sigma = ([\sigma_1]_1, [\sigma_2]_2, [A_i(x)]_1, [B_i(x)]_1, [B_i(x)]_2, [C_i(x)]_1) \tag{1.14}
$$

4. Destroy the toxic waste τ

5. Return σ

1.3 Prove and verify

The prover receives as input the circuit information from QAP $A_i(X), B_i(X), C_i(X)$, the public input $a_1, a_2, \ldots a_l$, the CRS σ. The goal of the prover is to prove that he knows a witness $a_{l+1}, \ldots a_m$ such that (1.7) is satisfied. We outline the steps, although in an implementation these can be done in different ways. The goal here is to merely understand the steps involved. We will add a later section on how exactly to compute the various steps using FFT techniques

1. Choose random $r, s \in \mathbb{F}_r$

2. Using the elements in the CRS compute

$$
\begin{aligned}
[\pi_A]_1 &= [\alpha]_1 + \sum_{i=0}^{m} a_i \cdot [A_i(x)]_1 + r \cdot [\delta]_1 \\
[\pi_B]_2 &= [\beta]_2 + \sum_{i=0}^{m} a_i \cdot [B_i(x)]_2 + s \cdot [\delta]_2 \\
[B]_1 &= [\beta]_1 + \sum_{i=0}^{m} a_i \cdot [B_i(x)]_1 + s \cdot [\delta]_1 \\
[\pi_C]_1 &= \sum_{i=l+1}^{m} a_i \cdot [pk'_i]_1 + \sum_{i=0}^{n-2} h_i \cdot \left[\frac{x^i t(x)}{\delta}\right]_1 + s \cdot [\pi_A]_1 + r[B]_1 - r \cdot s[\delta]_1
\end{aligned}
\tag{1.15}
$$

where h_i are coefficients of the quotient polynomial $h(X) = \sum_{i=0}^{n-2} h_i X^i$

3. Send proof $\pi = ([A]_1, [C]_1, [B]_2)$

In the above equation, the group elements $[\alpha]_1, [\beta]_1$ are added to the proof for consistency in the pairing equation. The purpose of r, s is to add zero knowledge such that the witness is masked. We discuss this in detail in the appendix §**??**

The verifier recieves the circuit information from QAP $A_i(X), B_i(X), C_i(X)$, the public input $a_1, a_2, \dots a_l$, the CRS σ and the prover's message π. The verifier accepts the proof iff the following condition is satisfied

$$e([\pi_A]_1, [\pi_B]_2) = e([\alpha]_1, [\beta]_2) + e(\sum_{i=0}^{l} a_i [vk'_i]_1, [\gamma]_2) + e([\pi_C]_1, [\delta]_2) \tag{1.16}$$

The proof of the pairing equation is given in the appendix §A.1.

2 Algorithm for prover in Lagrange basis

In this section we give an algorithm for computing the proof where the quotient polynomial is computed in Lagrange basis

$$\begin{aligned}
QAP &= \{A_i(X), B_i(X), C_i(X), t(X)\} \ \forall i = 0, 1 \dots, m \\
CRS &= \Bigg\{ ([A_i(x)]_1, [B_i(x)]_1, [B_i(x)]_2, [C_i(x)]_1, \\
&\quad , [\alpha]_1, [\beta]_1, [\delta]_1, \left\{ \left[\frac{x^i t(x)}{\delta} \right]_1 \right\}_{i=0}^{n-2}, \{[vk'_i]_1\}_{i=0}^{m}, \{[pk'_i]_1\}_{i=0}^{m} \\
&\quad , [\beta]_2, [\gamma]_2, [\delta]_2, \{[x^i]_2\}_{i=0}^{n-1} \Bigg\} \\
[pk'_i]_1 &= \left\{ \left[\frac{\beta \cdot A_i(x) + \alpha \cdot B_i(x) + C_i(x)}{\delta} \right]_1 \right\}_{i=l+1}^{m} \\
[vk'_i]_1 &= \left\{ \left[\frac{\beta \cdot A_i(x) + \alpha \cdot B_i(x) + C_i(x)}{\gamma} \right]_1 \right\}_{i=0}^{l}
\end{aligned} \tag{2.1}$$

It is important for this algorithm that all the setup polynomials are computed in the lagrange basis. We will use the notation $\mu(X) = FFT_{n-1}\left(\left\{ \left[\frac{x^i t(x)}{\delta} \right]_1 \right\}_{i=0}^{n-2} \right)$. The algorithm is summarized in fig 2.

2.1 Computing the quotient polynomial and its commitment

In fig 2 we outlined the , quotient polynomial computation. Lets dive a bit deeper into the expression $h(x) = N(x)/t(x)$

$$N(x) = A(x) \cdot B(x) - C(x) \tag{2.2}$$

From (1.7) the degree of the quotient is $n-2$, i.e it needs $n-1$ samples to be well defined. Given a domain $H : \{1, \omega, \omega^2, \dots \omega^{n-1}\}$ and a coset domain with a generator $\eta \in \mathbb{F}$. In general η can be any $k > n$th root of unity for eg $\omega_{2n} = \sqrt{\omega_n}$ and we can define $2n$ points of evaluation in two distinct sets

$$H_0 = H \ , \ H_1 = \omega_{2n} \times H \tag{2.3}$$

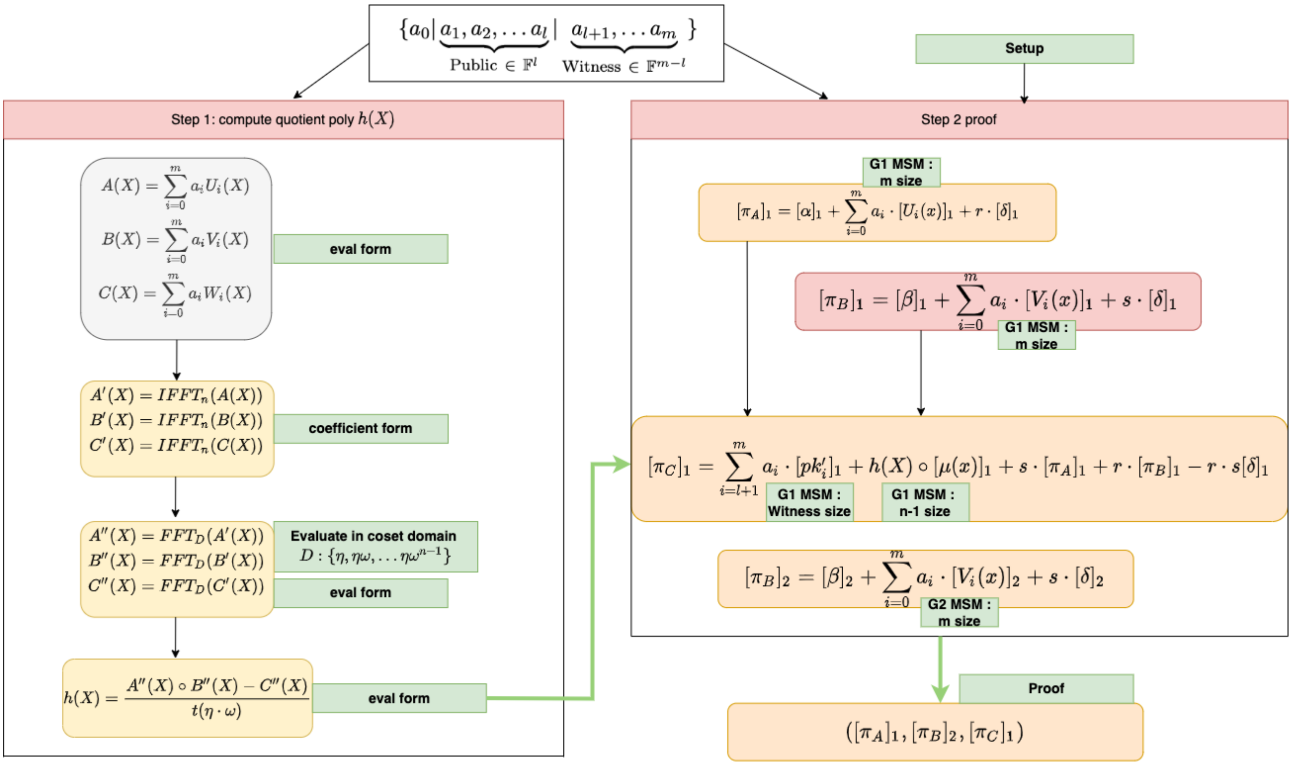

Figure 2. Groth16: Lagrange basis algorithm for prover

The advantage of this being NTTs can be defined for fixed sizes and parallelized across cosets. For example, for the product $A(x) * B(x)$, we can compute the result which needs $2n$ points in evaluation form as

$$A(x)|_{H_0} = FFT_{H_0}(A(x)) \tag{2.4}$$
$$A(x)|_{H_1} = FFT_{H_1}(A(x))) \tag{2.5}$$
$$B(x)|_{H_0} = FFT_{H_0}(B(x)) \tag{2.6}$$
$$B(x)|_{H_1} = FFT_{H_1}(B(x))) \tag{2.7}$$

Note that all the coset FFTs can be done simultaneously, since they are independent. Moreover one can compute the product in evaluation form as

$$\text{Canonical-order}\,[A(x)|_{H_0} \circ B(x)|_{H_0} || A(x)|_{H_1} \circ B(x)|_{H_1}] \tag{2.8}$$

where the canonical order step is required to arrange the samples into the correct order of evaluations for $2n$ points

$$[A(1)B(1), A(\omega^{1/2})B(\omega^{1/2}), A(\omega)B(\omega), \dots, A(\omega^{n-1/2})B(\omega^{n-1/2}), A(\omega^{n-1})B(\omega^{n-1})] \tag{2.9}$$

Here there are two equivalent ways to compute the coset FFT, one is as shown above by defining coset domain. The other is to do do an order n multiplication on the coefficients and keep the same domain.

$$FFT_{H_\eta}(A(x)) = FFT_H(A(\eta x)) \tag{2.10}$$

Note that in both approaches there is a slight tradeoff, which may be relevant depending on the application and use case specific to devices such as GPU. In the coset domain approach, the compute complexity is $\mathcal{O}(n \log n)$, however it is difficult to work with on device domain. Since each time an NTT kernel is launched for eg in a GPU the appropriate coset domain must be loaded on device. So this is preferrable when the number of cosets are small to avoid increasing host-device communication complexity.

On the other hand, if one chooses to work with only the base domain on device in memory, one has to compute the vector $(1, \eta, \ldots, \eta^{n-1})$ in $\mathcal{O}(n)$ and $\mathcal{O}(n)$ multiplications from the hadamard product to the coefficients in $A(\eta x)$ This leads to a compute complexity of $\mathcal{O}(2n + n \log n)$. However, there is no need to switch domain, which reduces communication complexity

2.2 Can we get rid of NTTs and send them all to set up?

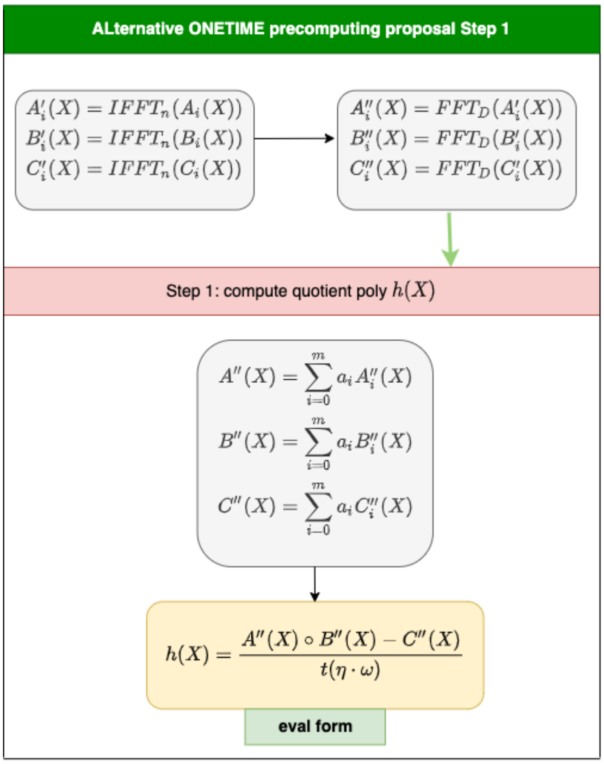

Figure 3. A potential optimization for step 1

Yes we can, and the price is exponential increase in compute so this is not preferable. However this is instructive. In fig 3 we describe an algorithm that moves all the FFT to preprocessing. It is very common that the R1CS matrices $A_i(X), B_i(X), C_i(X)$ are mostly sparse, the pre-processing step on the R1CS

$$\begin{aligned} A_i''(X) &= FFT_D(IFFT_n(A_i(X)) \\ B_i''(X) &= FFT_D(IFFT_n(B_i(X)) \\ C_i''(X) &= FFT_D(IFFT_n(C_i(X)) \end{aligned} \tag{2.11}$$

renders $A_i''(X), B_i''(X), C_i''(X)$ into a dense non-sparse matrices. The complexity of the subsequent step

$$\begin{aligned} A(X) &= \sum_{i=0}^{m} a_i \cdot A_i''(X) \\ B(X) &= \sum_{i=0}^{m} a_i \cdot B_i''(X) \\ C(X) &= \sum_{i=0}^{m} a_i \cdot C_i''(X) \end{aligned} \tag{2.12}$$

of the order $3 \cdot \mathcal{O}(n \cdot m)$ whereas the subsequent complexity in the regular groth 16 before the $h(X)$ computation is $\mathcal{O}(s_a + s_b + s_c) \cdot m + 6 \cdot n \cdot \log_2 n$, where s_a, s_b, s_c are the number of non sparse entries in the A, B, C matrices respectively. Thus the new method is better only if

$$3 \cdot n \cdot m < (s_a + s_b + s_c) \cdot m + 6 \cdot n \cdot \log_2 n \tag{2.13}$$

Typically $m \sim \mathcal{O}(n)$ and the worst case non-sparsity is $\mathcal{O}(100)$ thus we need

$$n < 100 + 2 \cdot \log_2 n \tag{2.14}$$

which is only true as it is for $n < 2^6$. However one can perhaps parallelize the computation. If one assumes that the complexity drops by a factor of k, where we assume that there are k threads in a GPU. For typical sizes $n = 2^{17}$, the number of threads is $\sim 2^{11}$ for the method to be effective in terms of complexity reduction. This implies that this method is unsuitable for large n. The second issue that comes up is the fact that the non sparse R1CS matrices need to be stored in memory, else the load time is significantly slow, compared to the compute time. If one would like to store the matrices in the GPU memory apriori it still costs significant memory (petabytes) for $n, m \sim 2^{27}$.

2.3 Another hack to avoid division by vanishing polynomial

This is due to [4]. We note in (1.15) that the term in π_c (coefficients form) where h_i is tje result after dividing by the vanishing polynomial $t(x)$

$$h_i \cdot \left[\frac{x^i t(x)}{\delta}\right]_1 \tag{2.15}$$

In Lagrange form, before dividing by the vanishing polynomial it looks like

$$\frac{N(x)\big|_{H_\eta}}{t(x)|_{H_\eta}} \cdot [\mu(x)]_1 \tag{2.16}$$

where $N(x) = A(x) \cdot B(x) - C(x)$ and $[\mu(x)]_1 = FFT_{n-1}\left(\left\{\left[\frac{x^i t(x)}{\delta}\right]_1\right\}_{i=0}^{n-2}\right)$ represents the corresponding CRS terms in Lagrange basis. Since

$$(t(x)\big|_{H_\eta})^{-1} = (\eta^n - 1)^{-1}[1, 1, 1, \ldots, 1] \tag{2.17}$$

one can precompute this and redefine

$$[\tilde{\mu}(x)]_1 = (t(x)\big|_{H_\eta})^{-1} \cdot [\mu(x)]_1 \tag{2.18}$$

which can be computed using an multi scalar multiplication. Thus reducing the prover computation of the quotient computation to

$$N(x) \cdot [\tilde{\mu}(x)]_1 \tag{2.19}$$

just computing the numerator on the coset and committing the result. This works because the relevant terms in the pairing equation §A.1 cancel only if the constraints $N(x)$ are satisfied in the domain.

3 Vulnerabilities in groth16

To update

4 Proof aggregation

To update

A Appendix

A.1 Proof of pairing equation (1.16)

The pairing equation is given by (1.16)

$$e([A]_1, [B]_2) = e([\alpha]_1, [\beta]_2) + e(\sum_{i=0}^{l} a_i [vk_i']_1, [\gamma]_2) + e([C]_1, [\delta]_2) \tag{A.1}$$

below we prove the above equation. We will use the properties that the pairing satisfies the bilinear maps property

$$\begin{aligned} e([P + P]_1', [Q]_2) &= e([P]_1, [Q]_2) + e([P']_1, [Q]_2) \\ e([P]_1, [Q + Q']_2) &= e([P]_1, [Q]_2) + e([P']_1, [Q']_2) \\ e(a \cdot [P]_1, b \cdot [Q]_2) &= e([P]_1, b \cdot [Q]_2)^a = e(a \cdot [P]_1, [Q]_2)^b \\ &= e([P]_1, [Q]_2)^{a \cdot b} = e(b \cdot [P]_1, a \cdot [Q]_2) \end{aligned} \tag{A.2}$$

Using (1.15), we expand (1.16) to

$$
\begin{aligned}
&e\left([\alpha]_1,[\beta]_2\right)+e\left([\alpha]_1,\sum_{i=0}^{m}a_i[B_i(x)]_2\right)+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,[\beta]_2\right)+e\left([\alpha]_1,s[\delta]_2\right)\\
&+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,\sum_{i=0}^{m}a_i[B_i(x)]_2\right)+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,s[\delta]_2\right)+e\left(r[\delta]_1,[\beta]_2\right)+e\left(r[\delta]_1,\sum_{i=0}^{m}a_i[B_i(x)]_2\right)\\
&+e\left(r[\delta]_1,s[\delta]_2\right)\\
&\overset{?}{=}\\
&e\left(\sum_{i=l+1}^{m}a_i[pk'_i]_1,[\delta]_2\right)+e\left(\sum_{i=0}^{n-2}h_i\left[\frac{x^it(x)}{\delta}\right]_1,[\delta]_2\right)+e\left([\alpha]_1,s[\delta]_2\right)+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,s[\delta]_2\right)\\
&+e(r[\delta]_1,s[\delta]_2)+e([\beta]_1,r[\delta]_2)+e\left(\sum_{i=0}^{m}a_i[B_i(x)]_1,r[\delta]_2\right)+e([\alpha]_1,[\beta]_2)+e\left(\sum_{i=0}^{l}a_i[vk'_i]_1,[\gamma]_2\right)
\end{aligned}
\tag{A.3}
$$

The terms in red, the terms in blue and green are to be identified with their corresponding term in the RHS due to bilinearity. For eg since we know that $\delta.\beta = \delta\beta$

$$
e([\delta]_1,[\beta]_2)=e([\beta]_1,[\delta]_2)=[\delta\beta]_T \tag{A.4}
$$

since δ,β commute and there is additive homomorphism, these terms should be identical in a valid proof. The terms in black in the RHS (A.3) can be written as follows

$$
\begin{aligned}
e\left(\sum_{i=l+1}^{m}a_i[pk'_i]_1,[\delta]_2\right)+e\left(\sum_{i=0}^{l}a_i[vk'_i]_1,[\gamma]_2\right)&=e\left(\sum_{i=l+1}^{m}a_i\left[\frac{\beta A_i(x)+\alpha B_i(x)+C_i(x)}{\delta}\right]_1,[\delta]_2\right)\\
&\quad+e\left(\sum_{i=0}^{l}a_i\left[\frac{\beta A_i(x)+\alpha B_i(x)+C_i(x)}{\gamma}\right]_1,[\gamma]_2\right)\\
&=e\left(\sum_{i=l+1}^{m}a_i\left[\beta A_i(x)+\alpha B_i(x)+C_i(x)\right]_1,[1]_2\right)\\
&\quad+e\left(\sum_{i=0}^{l}a_i\left[\beta A_i(x)+\alpha B_i(x)+C_i(x)\right]_1,[1]_2\right)\\
&=e\left(\sum_{i=0}^{m}a_i\left[\beta A_i(x)+\alpha B_i(x)+C_i(x)\right]_1,[1]_2\right)
\end{aligned}
\tag{A.5}
$$

Thus we are left to show

$$
\begin{aligned}
&e\left([\alpha]_1,\sum_{i=0}^{m}a_i[B_i(x)]_2\right)+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,[\beta]_2\right)+e\left(\sum_{i=0}^{m}a_i[A_i(x)]_1,\sum_{i=0}^{m}a_i[B_i(x)]_2\right)\\
&\overset{?}{=}\\
&e\left(\sum_{i=0}^{n-2}h_i\left[\frac{x^it(x)}{\delta}\right]_1,[\delta]_2\right)+e\left(\left(\sum_{i=0}^{m}a_i\left[\beta A_i(x)+\alpha B_i(x)+C_i(x)\right]_1\right),[1]_2\right)
\end{aligned}
\tag{A.6}
$$

we then use

$$e\left([\alpha]_1, \sum_{i=0}^{m} a_i[B_i(x)]_2\right) + e\left(\sum_{i=0}^{m} a_i[A_i(x)]_1, [\beta]_2\right) = e\left(\sum_{i=0}^{m}\ a_i[\alpha\ B_i(x)]_1, [1]_2\right) + e\left(\sum_{i=0}^{m} a_i[\beta\ A_i(x)]_1, [1]_2\right)$$
$$= e\left(\sum_{i=0}^{m} a_i\left[\beta A_i(x) + \alpha B_i(x) + C_i(x)\right]_1, [1]_2\right) \tag{A.7}$$

substituting the above in (A.6) and simplifying we get

$$e\left(\sum_{i=0}^{m} a_i[A_i(x)]_1, \sum_{i=0}^{m} a_i[B_i(x)]_2\right) \stackrel{?}{=} e\left(\sum_{i=0}^{n-2} h_i\left[x^i t(x)\right]_1, [1]_2\right) + e\left(\sum_{i=0}^{m} a_i\left[C_i(x)\right]_1, [1]_2\right) \tag{A.8}$$

which basically holds if there exists some witness in a_i that satisfies (1.7).

B Definitions

Acknowledgments

We stand on the shoulders of giants.

References

[1] J. Groth, "On the size of pairing-based non-interactive arguments." Cryptology ePrint Archive, Paper 2016/260, 2016.

[2] ZKproofs, "Setup ceremonies." https://zkproof.org/2021/06/30/setup-ceremonies/.

[3] B. Eddington, "Bls12-381 for the rest of us." https://hackmd.io/@benjaminion/bls12-381#Pairings.

[4] Iden3, "Snarkjs." https://github.com/iden3/snarkjs.