A R1CS Gadget for a 2^k-bit Scaling of a Fixed Generator in 7 Multiplicative Constraints

Luke "Kayaba" Parker

May 8, 2024

Abstract

Scalar multiplications are frequently performed within arithmetic circuits. Due to their expense, we specify an efficient protocol achieving a constant amount of multiplicative constraints.

1 Background

Scalar multiplications are common for re-randomizing values, such as Pedersen Commitments, within privacy protocols and for verifying smaller proofs within aggregating succinct proofs. For a 256-bit elliptic curve, the leading method uses 512 muliplicative constraints for a scalar already decomposed to bits (with such a decomposition taking an additional 256 multiplicative constraints). This involves using 128 multiplicative-constraints for the logical AND of each pair of bits, then using them in 128 2-bit lookups, finally performing 128 incomplete additions (each taking 3 multiplicative constraints).

This work builds on Eagen's preprint, https://eprint.iacr.org/2022/596, to posit a gadget using just 7 multiplicative constraints.

2 Notation

E is an elliptic curve over a prime field F (though not necessarily of prime order) with a curve equation of $y^2 = x^3 + a * x + b$. Points on the elliptic curve are written with capital letters (T, U), identity represented by 0. The group operation is written additively, scaling multiplicatively.

3 Line Function

$\ell(A, B)$ represents the line function which returns a polynomial $\mod y^2 - x^3 - a * x - b$ with variables x, y. If $A == -B$, the line function returns $x - \mathsf{A.x}$. If $A == B$, the function continues with $B = -(A + A)$. The function returns $y - (\delta * x) - \mu$ where $\delta = (\mathsf{B.y} - \mathsf{A.y}) * (\mathsf{B.x} - \mathsf{A.x})^{-1}$ and $\mu = \mathsf{A.y} - (\delta * \mathsf{A.x})$.

4 An Interactive Protocol for Proving Sums of Points

$$\{\ P \in E^n;\ \_ \mid \sum_{i=0}^{n} P = 0\ \}$$

We specify a sound interactive protocol for this statement.
Please note the _ represents how the witness is empty.

1. The prover calculates:

   $l_0 = \ell(P_0, -P_0)$

   $\forall\ 1 <= i < n : l_i = \ell(\sum_{j=0}^{i-1} P_j,\ P_i)$

   $d = \prod_{i=0}^{n} l_i / \prod_{i=0}^{n-1} \ell(\sum_{j=0}^{i} P_j,\ -\sum_{j=0}^{i} P_j)$

   d is decomposed into the zero-indexing $d_y, d_{yx_i}, d_{x_i}, d_0$. Since the polynomial is $\mod y^2 - x^3 - a * x - b$, there will only be a single d_y coefficient (hence the lack of indexing) and all d_{yx} coefficients will only be for $y^1 * x^i$ (hence the singular index). d_0 represents the coefficient for $x^0 * y^0$.

   d is scaled in-place by the inverse of $d_{x,0}$ such that $d_{x,0}$ becomes 1.

   The prover sends $d_y, d_{yx_{0..m}}, d_{x_{1..o}}, d_0$ to the verifier, where m, j are the lengths of d_{yx}, d_x respectively.

2. The verifier defines $d_{x_0} = 1$. They then differentiate the polynomial d by y (obtaining e) and by x (obtaining f).

   $e_y = 0$

   $e_{yx} = [\ ]$

   $e_x = d_{yx}$

   $e_0 = d_y$

   $f_y = d_{yx_0}$

   $f_{yx} = [\forall\ 1 <= i < m : (i+1) * d_{yx_i}]$

   $f_x = [\forall\ 1 <= i < o : (i+1) * d_{x_i}]$

   $f_0 = d_{x_0}$

3. The verifier choose two random challenge points, A_0, A_1 and performs the following:

   $A_2 = -(A_0 + A_1)$

   $\delta = (A_1.\mathsf{y} - A_0.\mathsf{y}) * (A_1.\mathsf{x} - A_0.\mathsf{x})^{-1}$

   $\mu = A_0.\mathsf{y} - (slope * A_0.\mathsf{x})$

   $l = \sum_{i=0}^{2} \frac{(\frac{3*A_i.\mathsf{x}^2+a}{2*A_i.\mathsf{y}} * f(\mathsf{x}=\mathsf{A_i}.\mathsf{x},\mathsf{y}=\mathsf{A_i}.\mathsf{y})) + e(\mathsf{x}=\mathsf{A_i}.\mathsf{x},\mathsf{y}=\mathsf{A_i}.\mathsf{y})}{d(\mathsf{x}=\mathsf{A_i}.\mathsf{x},\mathsf{y}=\mathsf{A_i}.\mathsf{y})} * \frac{2*A_i\mathsf{y}}{(-\delta*(2*A_i.\mathsf{y})) + (3*A_i.\mathsf{x}^2+a)}$

   $r = \sum_{i=0}^{n} (\mu - (P_i.\mathsf{y} + (\delta * P_i.\mathsf{x})))^{-1}$

4. The verifier checks $l = r$.

4.1 Soundness

We prove soundness for the protocol via witness extraction. Given the witness is _, we replace the statement with the following.

$$\{P \in E^n;\ l \in F^n \mid l_0 = \ell(P_0,\ -P_0) \wedge \forall\ 1 <= i < n : l_i = \ell(\sum_{j=0}^{i-1} P_j, P_i) \wedge l_{n-1} = x - P_{n-1}.\mathsf{x}\ \}$$

We proceed extract the lines from the transcript of the function.
TODO

4.2 Proof of Scalar Multiplication

The following proof naturally enables proving scalar multiplications via double-and-add.

$$\{\ G, P \in E, s \in F^k \mid \sum_{i=0}^{k} P = s_i \cdot (2^i \cdot G)\ \}$$

Please note s may be a non-canonical representation of the scalar. Assuming $G, H \in E$ of the same prime order, a proof for $(s,\ G = G)$ and a proof for $(s,\ G = H)$ will prove for a consistent scalar however.

1. The prover and verifier calculate $\forall\ 0 <= i < k :\ G_i = 2^i \cdot G$.
2. The prover and verifier perform the sum-proof protocol where the summed list contains $-P$ and s_i instances of G_i (for a list length of $1 + \sum_{i=0}^{n} s_i$).

This allows proving a k-bit scalar multiplication with a $1 + k$ length list. A prover who malleates s from a bitstring will have a longer list/divisor, yet the verifier may bound m, o since there's an upper-bound on the necessary length of the list for a k-bit scalar.

5 Arithmetization

We assume an arithmetic circuit with two parts to its statement.

- $a_l * a_r = a_o$

  For vectors a_l, a_r, a_o, a_o is the Hadamard product of a_l, a_r.

- $W_l * a_l + W_r * a_r + W_o * a_o + W_v * V + c = 0$

  W_l, W_r, W_o, W_v, c have an equal amount of columns. An instance of this formula, referred to as a constraint, exists per column. Within an instance, we refer to W_l, W_r, W_o, W_v, c as the column relevant (and not the overall structure).

  W_l, W_r, W_o, W_V has as many rows as a_l, a_r, a_o. W_v additionally has a depth equivalent to the amount of Pedersen vector commitments. c is a single value.

  $W_{l,r,o} * a_{l,r,o}$ is $\sum_{i=0}^{a_l.\mathsf{len}()} W_{l,r,o_i} * a_{l,r,o_i}$ (where l, r, o refers to one of the relevant three).

  $W_v * V$ is $\sum_{i=0}^{V.\mathsf{len}()} \sum_{j=0}^{a_l.\mathsf{len}()} W_{v_{i,j}} * V_{i,j}$

  This is almost identical to the arithmetic circuit statement from Bulletproofs(+), barring c being on the left-hand side. Conversion is as simple as subtracting c from both sides for each constraint.

A gadget is a series of a_l, a_r (and respective a_o) rows and columns in W_l, W_r, W_o, W_v, c. They are intended to be reusable snippets we can independently formally verify as correct/create security proofs for.

We define $a_l.\mathsf{push}, a_r.\mathsf{push}, V_i\emptyset$ as functions pushing a linear combination or value onto the respective vector, returning a constrainable reference. Once $a_l.\mathsf{push}, a_r.\mathsf{push}$ have been called, the push to a_o is implicit with a constrainable reference retrievable with $\mathsf{a_o}.\mathsf{last}()$.

5.1 Gadgets

5.1.1 Equality

$equality(a, b) \rightarrow$
$1\ a + -1\ b = 0$
By adding b to both sides, we get $a = b$.

5.1.2 Inverse

$\mathsf{Inverse}(a) : z \rightarrow$
$\mathsf{Equality}(a_l.\mathsf{push}(a),\ a)$
$z = a_r.\mathsf{push}(a^{-1})$
$1\mathsf{a_o}.\mathsf{last}() + -1 = 0$
$\mathsf{return}\ z$
For $a * z = 1$, we can rewrite this as $z = 1/a$ (which is the multiplicative inverse of a).

5.1.3 On Curve

$\mathsf{OnCurve}_{\mathsf{a,b}}(x, y) \rightarrow$
$\mathsf{Equality}(a_l.\mathsf{push}(x), x)$
$\mathsf{Equality}(a_r.\mathsf{push}(x), x)$
$x^2 = \mathsf{a_o}.\mathsf{last}()$
$\mathsf{Equality}(a_l.\mathsf{push}(x^2), x^2)$
$\mathsf{Equality}(a_r.\mathsf{push}(x), x)$
$x^3 = \mathsf{a_o}.\mathsf{last}()$
$\mathsf{Equality}(a_l.\mathsf{push}(y), y)$
$\mathsf{Equality}(a_r.\mathsf{push}(y), y)$
$y^2 = \mathsf{a_o}.\mathsf{last}()$
$\mathsf{Equality}(1\ y^2, 1\ x^3 + a\ x + 1\ b)$

5.1.4 Discrete Log Gadget

We evaluate the proof of scalar multiplication within a gadget. We assume a pair of challenges $\mathsf{chl}_0, \mathsf{chl}_1$ properly derived from the statement/transcript. In order to form the transcript, the arithmetic circuit proof ideally supports an extra round of interactivity or vector commitments. Naively, any proof capable of opening Pedersen commitments may be used (by placing all arguments to the gadgets within the commitments and then hashing them).

$\mathsf{DivisorChallenge}_{\mathsf{a,b}}(d_y, d_{yx_{0..m}}, d_{x_{1..o}}, d_0, \delta, \mathsf{chl}) \rightarrow$
$p_{0_{n_0}} = (3 * \mathsf{chl.x}^2 + a)/(2 * \mathsf{chl.y})$
$p_{0_{n_1}} = \sum_{j=0}^{m}(p_{0_{n_0}} * \mathsf{chl.x}^{j+1}) * d_{yx_j} + (p_{0_{n_0}} * d_y)$
$p_{0_{n_2}} = \mathsf{chl.y} * d_{yx_0} + \sum_{j=1}^{m}((j+1) * \mathsf{chl.y} * \mathsf{chl.x}^j) * d_{yx_j} + \sum_{i=1}^{o}((i+1) * \mathsf{chl.x}^i) * d_{x_i} + 1$
$p_{0_n} = p_{0_{n_1}} + p_{0_{n_2}}$
$p_{0_d} = \mathsf{chl.y} * d_y + \sum_{i=0}^{m}(\mathsf{chl.y} * \mathsf{chl.x}^{(i+1)}) * d_{xy_i} + \sum_{i=1}^{o} \mathsf{chl.x}^{(i+1)} * d_{x_i} + 1 * \mathsf{chl.x} + 1 * d_0$
$p_{1_n} = 2 * \mathsf{chl.y}$
$p_{1_d} = (-\delta \cdot p_{1_n}) + 3 * \mathsf{chl.x}^2 + a$
$p_n = p_{0_n} * p_{1_n}$
$p_d = p_{0_d} * p_{1_d}$
$\mathsf{Equality}(a_l.\mathsf{push}(p_d), p_d)$
$o = a_r.\mathsf{push}(p_n * p_d^{-1})$
$\mathsf{Equality}(\mathsf{a_o}.\mathsf{last}(), p_n)$
return o

While evaluating divisors (a polynomial), we consider the evaluation not as $a*x^2+b*x+c$ (a generic polynomial used for this example), yet $x^2*a+x*b+c$. The latter lines up with the constraint system's definition (since our x is public yet our coefficients are not).

Please note p_{1_n}, p_{1_d} are scalars, not linear combinations, despite our extensive usage of linear combinations here.

$\mathsf{DiscreteLog}_G(s, x, y, d_y, d_{yx}, d_x, d_0) \rightarrow$
$\mathsf{OnCurve}_{\mathsf{a,b}}(x, y)$
$\mathsf{chl}_0 = \mathcal{H}_{point}(\mathsf{chl}_0)$
$\mathsf{chl}_1 = \mathcal{H}_{point}(\mathsf{chl}_1)$
$\mathsf{chl}_2 = -(\mathsf{chl}_0 + \mathsf{chl}_1)$
$\delta = (\mathsf{chl}_1.\mathsf{y} - \mathsf{chl}_0.\mathsf{y}) * (\mathsf{chl}_1.\mathsf{x} - \mathsf{chl}_0.\mathsf{x})^{-1}$
$\mu = \mathsf{chl}_0.\mathsf{y} - (\delta * \mathsf{chl}_0.\mathsf{x})$
$f = \mathsf{Inverse}(\mu - (-y + (\delta * \ x)))$
$\mathsf{Equality}(\sum_{i=0}^{2} \mathsf{DivisorChallenge}_{\mathsf{a,b}}(d_y, d_{yx}, d_x, d_0, \delta, \mathsf{chl}_i), \sum_{i=0}^{k}(\mu - (G_i.\mathsf{y} + (\delta * G_i.\mathsf{x})))^{-1} * s_i + 1\ f)$

For a fixed generator G, the prover and the verifier calculate $\forall\ 0 <= i < k:\ G_i = 2^i \cdot G$ (as done in the interactive protocol).

Instead of performing the sum protocol with an expanded list, the relevant term of the right-hand side of the equality check is multiplied by the amount of intended inclusions. Since the terms of the right-hand side is part of a sum, multiplication by the amount of presences is equivalent to actually having that many instances present.

The on-curve gadget takes three multiplicative constants. The inverse gadget takes one multiplicative constraint. Finally, the three evaluations of $\mathsf{DivisorChallenge}_{\mathsf{a,b}}$ each take one multiplicative constraint. In total, the gadget takes seven, regardless of k.