Fast elliptic curve scalar multiplications in SN(T)ARK circuits

Youssef El Housni (Linea), Thomas Piellard (Linea), Simon Masson (ZKNox) and Liam Eagen (Alpen Labs)

October 1st, 2025 – *Medellin, Latincrypt*

consensys

Linea

Overview

1. Preliminaries
   - SNARKs
   - Elliptic curves
2. Contributions
   - Elliptic curves in SNARKs
   - Implementation

Preliminaries

(Zero-knowledge) Succinct Non-interactive ARguments of Knowledge (zk-SNARK)

Let F be a public NP program, x and z be public inputs, and w be a private input such that

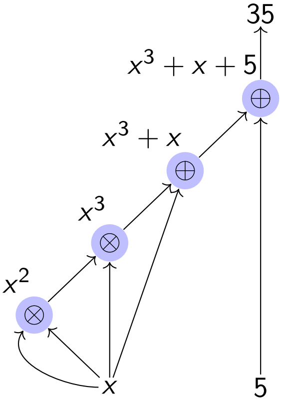
$$z := F(x, w)$$

.

A ZK-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

Setup:	(pk, vk)	$\leftarrow$	$S(F, 1^\lambda)$
Prove:	π	$\leftarrow$	$P(x, z, w, pk)$
Verify:	`false/true`	$\leftarrow$	$V(x, z, \pi, vk)$

$$x^3 + x + 5 = 35 \qquad (x = 3)$$



constraints:

$$o = l \cdot r$$

$$a = x \cdot x$$
$$b = a \cdot x$$
$$c = (b + x) \cdot 1$$
$$d = (c + 5) \cdot 1$$

witness:

$$\vec{w} = \begin{pmatrix} \text{one} & x & d & a & b & c \end{pmatrix} = \begin{pmatrix} 1 & 3 & 35 & 9 & 27 & 30 \end{pmatrix}$$

SNARKs examples: Groth16 and PLONK

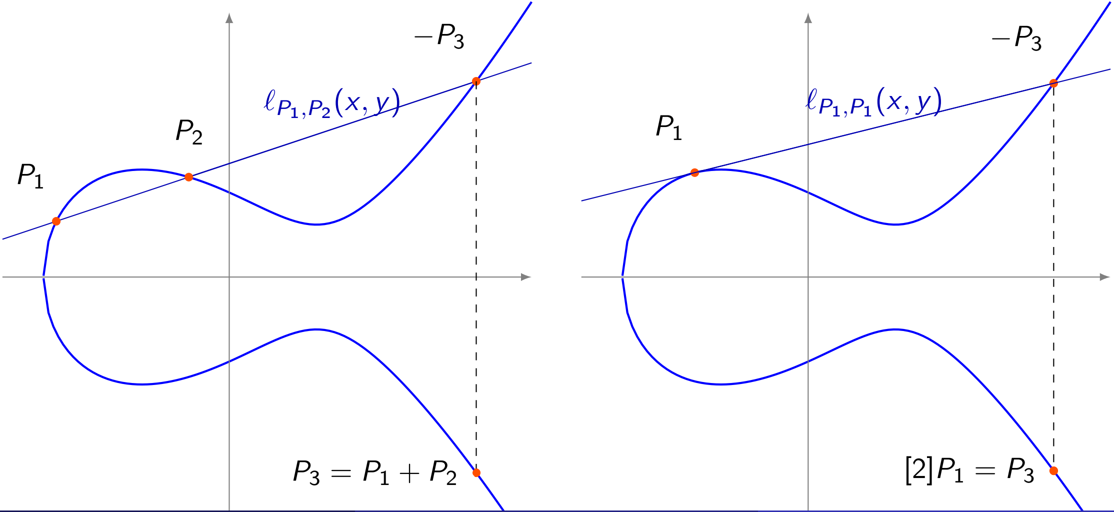
- m = number of wires
- n = number of multiplications gates
- a = number of additions gates
- ℓ = number of public inputs
- $\mathtt{M}_{\mathbb{G}}$ = multiplication in $\mathbb{G}$
- P=pairing

	Setup	Prove	Verify
Groth16 [Gro16]	$3n\ \mathtt{M}_{\mathbb{G}_1}$ $m\ \mathtt{M}_{\mathbb{G}_2}$	$(3n+m-\ell)\ \mathtt{M}_{\mathbb{G}_1}$ $n\ \mathtt{M}_{\mathbb{G}_2}$ 7 FFT	3P $\ell\ \mathtt{M}_{\mathbb{G}_1}$
PLONK (KZG) [GWC19]	$d_{\geq n+a}\ \mathtt{M}_{\mathbb{G}_1}$ $1\ \mathtt{M}_{\mathbb{G}_2}$ 8 FFT	$9(n+a)\ \mathtt{M}_{\mathbb{G}_1}$ 8 FFT	2P $18\ \mathtt{M}_{\mathbb{G}_1}$

Elliptic curves

$E\colon Y^2 = X^3 + aX + b$ elliptic curve defined over $\mathbb{F}_q$ and $r \mid \#E(\mathbb{F}_q)$

Figure: Chord-and-tangent rule over $\mathbb{R}$



Scalar Multiplication

1. How to compute $[141]P$?

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2]P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^3]P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^4]P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^4]P + P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2]([2^4]P + P)$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2]([2^4]P + P) + P$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2]([2]([2^4]P + P) + P)$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P)$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$
\begin{aligned}
& [10001101_2]P_1 \\
& +[10111000_2]P_2 \\
\text{Precomputed table :} \quad & \{0, P_1, P_2, P_1 + P_2\}
\end{aligned}
$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$
\begin{aligned}
& [10001101_2]P_1 \\
& +[10111000_2]P_2 \\
\text{Precomputed table :}\quad & \{0, P_1, P_2, P_1 + P_2\}
\end{aligned}
$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$\begin{aligned} & [10001101_2]P_1 \\ & +[10111000_2]P_2 \\ \text{Precomputed table :} \quad & \{0, P_1, P_2, P_1 + P_2\} \end{aligned}$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$
\begin{aligned}
& [10001101_2]P_1 \\
& +[10111000_2]P_2 \\
\text{Precomputed table}: \quad & \{0, P_1, P_2, P_1 + P_2\}
\end{aligned}
$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$\begin{aligned} & [10001101_2]P_1 \\ & +[10111000_2]P_2 \\ \text{Precomputed table}: \quad & \{0, P_1, P_2, P_1 + P_2\} \end{aligned}$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$
\begin{aligned}
& [10001101_2]P_1 \\
& +[10111000_2]P_2 \\
\text{Precomputed table}: \quad & \{0, P_1, P_2, P_1 + P_2\}
\end{aligned}
$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$\begin{aligned} &[10001101_2]P_1 \\ &+[10111000_2]P_2 \\ \text{Precomputed table : } \quad &\{0, P_1, P_2, P_1 + P_2\} \end{aligned}$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$[10001101_2]P_1$$
$$+[10111000_2]P_2$$
$$\text{Precomputed table}: \quad \{0, P_1, P_2, P_1 + P_2\}$$

Scalar Multiplication

1. How to compute $[141]P = [10001101_2]P$?
   $Q = [2^2]([2]([2^4]P + P) + P) + P = [141]P$ ✓
   Cost: $o(\log(k))$ additions ($\log(k) = 256$).
2. How to compute $[10001101_2]P_1 + [10111000_2]P_2$?

$$\begin{aligned} & [10001101_2]P_1 \\ & +[10111000_2]P_2 \\ \text{Precomputed table :} \quad & \{0, P_1, P_2, P_1 + P_2\} \end{aligned}$$

Cost: $o(\log(k))$ additions + precomputation table.

Scalar Multiplication

Gallant–Lambert–Vanstone: a technique to speed up scalar multiplication for specific curves: $[k]P = [k_1]P + [k_2]\psi(P)$ where $\psi(P)$ is easy to compute, and k_1, k_2 halved size.
Examples:

Example

$j = 0$ curves (e.g. SECP256K1, BN256, BLS12-381), $Y^2 = X^3 + b$
$\psi : E \to E$ defined by $(x, y) \mapsto (\omega x, y)$ (and $0_E \mapsto 0_E$) such that $\psi(P) = [\lambda]P$, where both ω and λ are cubic roots of unity in $\mathbb{F}_p$ and $\mathbb{F}_r$ respectively.

1. write k as $k_1 + \lambda k_2 \pmod r$
2. replace $[\lambda]P$ by $\psi(P)$ and compute $[k_1]P + [k_2]\psi(P)$

Contributions

Motivation

Proving scalar multiplications using SNARKs:

- SNARK composition (proof of proof): BN254, BLS12-381, BLS12-377/BW6-761, MNT4/6
- zero-knowledge virtual machines (zkVM): BN254, BLS12-381, SECP256K1
- Verkle trie (data structure for Ethereum): Bandersnatch, Jubjub
- Account abstraction (Ethereum): P-256, Ed25519

Hinted scalar multiplication

Consider the equation $[k]P = Q$.

Hinted scalar multiplication

Consider the equation $[k]P = Q$.
The scalar k can be decomposed as $k = x/z \bmod r$.

Consider the equation $[k]P = Q$.

The scalar k can be decomposed as $k = x/z \bmod r$.

$\{x - kz = 0 \bmod r\}$ is a lattice of dimension 2:

$$\begin{pmatrix} r & 0 \\ k & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & 1 \end{pmatrix}$$

Hinted scalar multiplication

Consider the equation $[k]P = Q$.
The scalar k can be decomposed as $k = x/z \bmod r$.
$\{x - kz = 0 \bmod r\}$ is a lattice of dimension 2:

$$\begin{pmatrix} r & 0 \\ k & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & 1 \end{pmatrix} \sim \begin{pmatrix} \square\square\square & \square\square\square \\ \square\square\square & \square\square\square \end{pmatrix}$$

Apply lattice reduction (like LLL) to find a short vectors.
Expected size $\sqrt{r}$.

Hinted scalar multiplication

Consider the equation $[k]P = Q$.
The scalar k can be decomposed as $k = x/z \bmod r$.
$\{x - kz = 0 \bmod r\}$ is a lattice of dimension 2:

$$\begin{pmatrix} r & 0 \\ k & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & 1 \end{pmatrix} \sim \begin{pmatrix} \square\square\square & \square\square\square \\ \square\square\square & \square\square\square \end{pmatrix}$$

Apply lattice reduction (like LLL) to find a short vectors.
Expected size $\sqrt{r}$.

$$[k]P = Q \iff [x]P - [z]Q = 0$$

Consider the equation $[k]P = Q$.
The scalar k can be decomposed as $k = x/z \bmod r$.
$\{x - kz = 0 \bmod r\}$ is a lattice of dimension 2:

$$\begin{pmatrix} r & 0 \\ k & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & 1 \end{pmatrix} \sim \begin{pmatrix} \square\square\square & \square\square\square \\ \square\square\square & \square\square\square \end{pmatrix}$$

Apply lattice reduction (like LLL) to find a short vectors.
Expected size $\sqrt{r}$.

$$[k]P = Q \iff [x]P - [z]Q = 0$$

- $[k]P = Q$: scalar of size 256,
- $[x]P - [z]Q = 0$: scalars of size 128. ✓

Hinted double scalar multiplication

Consider the equation $[k_1]P_1 + [k_2]P_2 = Q$.

Hinted double scalar multiplication

Consider the equation $[k_1]P_1 + [k_2]P_2 = Q$.
The scalars k_1, k_2 can be simultaneously decomposed as

$$k_1 = \frac{x_1}{z} \bmod r, \qquad k_2 = \frac{x_2}{z} \bmod r.$$

Hinted double scalar multiplication

Consider the equation $[k_1]P_1 + [k_2]P_2 = Q$.
The scalars k_1, k_2 can be simultaneously decomposed as

$$k_1 = \frac{x_1}{z} \bmod r, \qquad k_2 = \frac{x_2}{z} \bmod r.$$

$\{x_1 - k_1 z = 0 \bmod r \text{ and } x_2 - k_2 z = 0 \bmod r\}$ form a lattice of dimension 3:

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ k_1 & k_2 & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 & 0 \\ 0 & \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & \square\square\square\square\square\square & 1 \end{pmatrix}$$

Hinted double scalar multiplication

Consider the equation $[k_1]P_1 + [k_2]P_2 = Q$.
The scalars k_1, k_2 can be simultaneously decomposed as

$$k_1 = \frac{x_1}{z} \bmod r, \qquad k_2 = \frac{x_2}{z} \bmod r.$$

$\{x_1 - k_1 z = 0 \bmod r \text{ and } x_2 - k_2 z = 0 \bmod r\}$ form a lattice of dimension 3:

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ k_1 & k_2 & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 & 0 \\ 0 & \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & \square\square\square\square\square\square & 1 \end{pmatrix} \sim \begin{pmatrix} \square\square\square\square & \square\square\square\square & \square\square\square\square \\ \square\square\square\square & \square\square\square\square & \square\square\square\square \\ \square\square\square\square & \square\square\square\square & \square\square\square\square \end{pmatrix}$$

Apply lattice reduction (like LLL) to find a short vectors.
Expected size $\sqrt[3]{r^2}$.

Consider the equation $[k_1]P_1 + [k_2]P_2 = Q$.
The scalars k_1, k_2 can be simultaneously decomposed as

$$k_1 = \frac{x_1}{z} \bmod r, \qquad k_2 = \frac{x_2}{z} \bmod r.$$

$\{x_1 - k_1 z = 0 \bmod r \text{ and } x_2 - k_2 z = 0 \bmod r\}$ form a lattice of dimension 3:

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ k_1 & k_2 & 1 \end{pmatrix} = \begin{pmatrix} \square\square\square\square\square\square & 0 & 0 \\ 0 & \square\square\square\square\square\square & 0 \\ \square\square\square\square\square\square & \square\square\square\square\square\square & 1 \end{pmatrix} \sim \begin{pmatrix} \square\square\square\square & \square\square\square\square & \square\square\square\square \\ \square\square\square\square & \square\square\square\square & \square\square\square\square \\ \square\square\square\square & \square\square\square\square & \square\square\square\square \end{pmatrix}$$

Apply lattice reduction (like LLL) to find a short vectors.
Expected size $\sqrt[3]{r^2}$.

$$[k_1]P_1 + [k_2]P_2 = Q \iff [x_1]P_1 + [x_2]P_2 - [z]Q = 0$$

Triple scalar multiplication with scalars of 171 bits. ✓

GLV hinted scalar multiplication

GLV: a technique to speed up scalar multiplication for specific curves:
$[k]P = [k_1]P + [k_2]\psi(P)$ where $\psi(P)$ is easy to compute, and k_1, k_2 halved size.

GLV hinted scalar multiplication

GLV: a technique to speed up scalar multiplication for specific curves:
$[k]P = [k_1]P + [k_2]\psi(P)$ where $\psi(P)$ is easy to compute, and k_1, k_2 halved size.
GLV with hints: a fraction decompositions in $\mathbb{Z}[\lambda]$ where λ is an eigenvalue of ψ.

GLV hinted scalar multiplication

GLV: a technique to speed up scalar multiplication for specific curves:
$[k]P = [k_1]P + [k_2]\psi(P)$ where $\psi(P)$ is easy to compute, and k_1, k_2 halved size.
GLV with hints: a fraction decompositions in $\mathbb{Z}[\lambda]$ where λ is an eigenvalue of ψ.

Single scalar multiplication with GLV and hint

$$\begin{pmatrix} r & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 \\ k & 0 & 1 & 0 \\ 0 & 0 & -\lambda & 1 \end{pmatrix}$$

$$[k]P = Q$$
$$\Updownarrow$$
$$[x]P + [y]\psi(P) - [z]Q - [t]\psi(Q) = 0$$

Quadruple 64-bit scalar multiplication.

GLV hinted scalar multiplication

GLV: a technique to speed up scalar multiplication for specific curves:
$[k]P = [k_1]P + [k_2]\psi(P)$ where $\psi(P)$ is easy to compute, and k_1, k_2 halved size.
GLV with hints: a fraction decompositions in $\mathbb{Z}[\lambda]$ where λ is an eigenvalue of ψ.

Single scalar multiplication with GLV and hint

$$\begin{pmatrix} r & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 \\ k & 0 & 1 & 0 \\ 0 & 0 & -\lambda & 1 \end{pmatrix}$$

$$[k]P = Q$$
$$\Updownarrow$$
$$[x]P + [y]\psi(P) - [z]Q - [t]\psi(Q) = 0$$

Quadruple 64-bit scalar multiplication.

Double scalar multiplication with GLV and hint

$$\begin{pmatrix} r & 0 & 0 & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1 & 0 & 0 \\ k_1 & 0 & k_2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -\lambda & 1 \end{pmatrix}$$

$$[k_1]P_1 + [k_2]P_2$$
$$\Updownarrow$$
$$[x_1]P_1 + [y_1]\psi(P_1) + [x_2]P_2 + [y_2]\psi(P_2)$$
$$-[z]Q - [t]\psi(Q) = 0$$

Sextuple 86-bit scalar multiplication.

Practical results

Implementation in the `gnark` library with two proof systems:
Groth16 (R1CS) and PLONK (SCS).

Practical results

Implementation in the gnark library with two proof systems: Groth16 (R1CS) and PLONK (SCS).

Curve	Previous work	This work	Speed-up
BN254	381467 scs 78246 r1cs	220436 scs 59351 r1cs	42% 24%
BLS12-381	539973 scs 110928 r1cs	307045 scs 84508 r1cs	43% 24%
Secp256k1	385461 scs 78940 r1cs	223188 scs 60089 r1cs	42% 24%
P-256	612759 scs 157685 r1cs	294128 scs 78940 r1cs	52% 50%
Jubjub	5863 scs 3314 r1cs	4549 scs 2401 r1cs	22% 28%

Table: Implementation results for some curves.

Implementation in the `gnark` library with two proof systems: Groth16 (R1CS) and PLONK (SCS).

Curve	Previous work	This work	Speed-up
BN254	381467 scs 78246 r1cs	220436 scs 59351 r1cs	42% 24%
BLS12-381	539973 scs 110928 r1cs	307045 scs 84508 r1cs	43% 24%
Secp256k1	385461 scs 78940 r1cs	223188 scs 60089 r1cs	42% 24%
P-256	612759 scs 157685 r1cs	294128 scs 78940 r1cs	52% 50%
Jubjub	5863 scs 3314 r1cs	4549 scs 2401 r1cs	22% 28%

Table: Implementation results for some curves.

- The scalar decomposition is not optimal yet (`xgcd` vs `lll`),

Thank you

- **Paper**: `https://eprint.iacr.org/2025/933.pdf`
- **Implementation**: `https://github.com/yelhousni/scalarmul-in-snark`
- **Use-cases**: `https://github.com/consensys/gnark`
- **Contact**: `https://yelhousni.github.io`

References I

Jens Groth.
On the size of pairing-based non-interactive arguments.
In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru.
PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge.
Cryptology ePrint Archive, Report 2019/953, 2019.