

# Montgomery ladders compute pairings

Alessandro Sferlazza

joint work with: G. Pope, K. Reijnders, D. Robert, B. Smith

<https://eprint.iacr.org/2025/672>

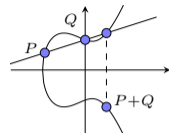
Technical University of Munich

Thursday 3 July 2025,  
GRACE seminar, Inria Saclay

# Main character: pairings on elliptic curves

Elliptic curves:  $E : y^2 = x^3 + ax + b$ , with  $a, b \in \mathbb{F}_q$

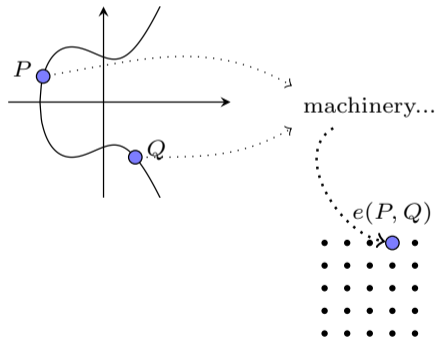
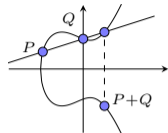
Points  $(x, y) \in \overline{\mathbb{F}}_q$  on the curve (+ a neutral element  $0_E$ ) form a group.



# Main character: pairings on elliptic curves

Elliptic curves:  $E : y^2 = x^3 + ax + b$ , with  $a, b \in \mathbb{F}_q$

Points  $(x, y) \in \overline{\mathbb{F}_q}$  on the curve (+ a neutral element  $0_E$ ) **form a group**.



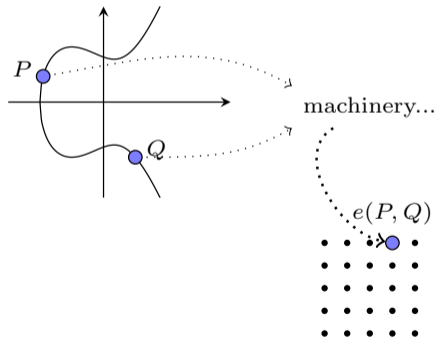
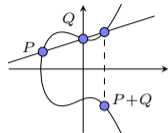
Pairings are maps from subgroups/quotients of some  $E$  to a finite field:

$$\begin{aligned} e_\ell : G_1 \times G_2 &\rightarrow G_T \subseteq \mathbb{F}_q^\times \\ (P, Q) &\mapsto e_\ell(P, Q) \end{aligned} \quad \ell \in \mathbb{N}$$

# Main character: pairings on elliptic curves

Elliptic curves:  $E : y^2 = x^3 + ax + b$ , with  $a, b \in \mathbb{F}_q$

Points  $(x, y) \in \overline{\mathbb{F}_q}$  on the curve (+ a neutral element  $0_E$ ) form a group.



Pairings are maps from subgroups/quotients of some  $E$  to a finite field:

$$\begin{aligned} e_\ell : G_1 \times G_2 &\rightarrow G_T \subseteq \mathbb{F}_q^\times \\ (P, Q) &\mapsto e_\ell(P, Q) \end{aligned} \quad \ell \in \mathbb{N}$$

They satisfy

- Bilinearity:  $e(P, Q + Q') = e(P, Q) \cdot e(P, Q')$
- Nondegeneracy: for all  $P \in G_1, Q \in G_2$  the maps  $e(P, \cdot)$  and  $e(\cdot, Q)$  aren't constantly trivial.
- ...and many other useful properties

## Pairings: usage and applications

Pairings: useful in different scenarios in cryptography.

Destructive use: transfer discrete logs on a curve  $E$  to easier discrete logs on  $\mathbb{F}_q^\times$

# Pairings: usage and applications

Pairings: useful in different scenarios in cryptography.

Destructive use: transfer discrete logs **on a curve  $E$**  to **easier** discrete logs **on  $\mathbb{F}_q^\times$**

Constructive use:

- advanced functionalities in encryption, signatures, ZK proofs...
  - ✓ Usually, freedom to choose base field  $\mathbb{F}_p$ , curve  $E \rightsquigarrow$  optimize for **fast arithmetic**

# Pairings: usage and applications

Pairings: useful in different scenarios in cryptography.

Destructive use: transfer discrete logs **on a curve  $E$**  to **easier** discrete logs **on  $\mathbb{F}_q^\times$**

Constructive use:

- advanced functionalities in encryption, signatures, ZK proofs...
    - ✓ Usually, freedom to choose base field  $\mathbb{F}_p$ , curve  $E \rightsquigarrow$  optimize for **fast arithmetic**
  - tool in **isogeny-based** cryptography. Here, **no control** over  $p, E$ :
    - ▶  $E$  usually a **random supersingular curve** over  $\mathbb{F}_{p^2}$ ,
    - ▶  $p$  subject to constraints  $\neq$  speed (namely,  $p + 1$  smooth for fast  $\mathbb{F}_{p^2}$ -isogenies)
- ✗ **fast arithmetic not always available.**

# Pairings: usage and applications

Pairings: useful in different scenarios in cryptography.

Destructive use: transfer discrete logs **on a curve  $E$**  to **easier** discrete logs **on  $\mathbb{F}_q^\times$**

Constructive use:

- advanced functionalities in encryption, signatures, ZK proofs...
    - ✓ Usually, freedom to choose base field  $\mathbb{F}_p$ , curve  $E \rightsquigarrow$  optimize for **fast arithmetic**
  - tool in **isogeny-based** cryptography. Here, **no control** over  $p, E$ :
    - ▶  $E$  usually a **random supersingular curve** over  $\mathbb{F}_{p^2}$ ,
    - ▶  $p$  subject to constraints  $\neq$  speed (namely,  $p + 1$  smooth for fast  $\mathbb{F}_{p^2}$ -isogenies)
- ✗ **fast arithmetic not always available.**
- $\rightsquigarrow$  **Need:** make **generic** pairings fast.

# Pairings: usage and applications

Pairings: useful in different scenarios in cryptography.

Destructive use: transfer discrete logs **on a curve  $E$**  to **easier** discrete logs **on  $\mathbb{F}_q^\times$**

Constructive use:

- advanced functionalities in encryption, signatures, ZK proofs...
    - ✓ Usually, freedom to choose base field  $\mathbb{F}_p$ , curve  $E \rightsquigarrow$  optimize for **fast arithmetic**
  - tool in **isogeny-based** cryptography. Here, **no control** over  $p, E$ :
    - ▶  $E$  usually a **random supersingular curve** over  $\mathbb{F}_{p^2}$ ,
    - ▶  $p$  subject to constraints  $\neq$  speed (namely,  $p+1$  smooth for fast  $\mathbb{F}_{p^2}$ -isogenies)
- ✗ **fast arithmetic not always available.**
- $\rightsquigarrow$  **Need:** make **generic** pairings fast.

generic $\ell$ -pairing: cost/bit	Tate pairing	Weil pairing
State of the art <sup>1</sup> using Miller's algo	11.3M + 7.7S + 20.7A	2 · Tate pairing
[Rob24] <sup>2</sup> $\rightsquigarrow$ our work	9M + 6S + 16A	

<sup>1</sup>Cai, Lin, Zhao, *Pairing Optimizations for Isogeny-based Cryptosystems*, eprint 2024/575

<sup>2</sup>Robert, *Fast pairings via biextensions and cubical arithmetic*, eprint2024/517

## Preliminaries: divisors

Divisors: Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

Divisors form a **group**. We focus on the subgroup of **divisors of degree 0**:

$$\text{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

## Preliminaries: divisors

Divisors: Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

Divisors form a **group**. We focus on the subgroup of **divisors of degree 0**:

$$\text{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

Principal divisors: Given  $f \in \overline{\mathbb{F}}_q(E)$ , we attach to it a **principal divisor**

$$\text{div } f = \sum_{P \in E} \text{ord}_P(f) \cdot (P)$$

where  $\text{ord}_P(f)$  is the multiplicity of  $P$  as a **zero** of  $f$  if  $> 0$ , and as **pole** of  $f$  if  $< 0$

## Preliminaries: divisors

Divisors: Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

Divisors form a **group**. We focus on the subgroup of **divisors of degree 0**:

$$\mathrm{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

Principal divisors: Given  $f \in \overline{\mathbb{F}}_q(E)$ , we attach to it a **principal divisor**

$$\mathrm{div} f = \sum_{P \in E} \mathrm{ord}_P(f) \cdot (P)$$

where  $\mathrm{ord}_P(f)$  is the multiplicity of  $P$  as a **zero** of  $f$  if  $> 0$ , and as **pole** of  $f$  if  $< 0$

Fact: Any  $E$  elliptic curve is **isomorphic** to a quotient of  $\mathrm{Div}^0(E)$ :

$$\begin{array}{ccc} E & \xrightarrow{\sim} & \mathrm{Pic}^0(E) \\ P & \mapsto & [(P) - (0_E)] \end{array} \quad = \mathrm{Div}^0(E) / \{\text{principal divisors}\}$$
$$\begin{array}{c} [D] = [D'] \\ \iff \\ D - D' = \mathrm{div} f \end{array}$$

# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{lll} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a [Miller function](#) attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a [Miller function](#) attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

✓ Other widely used pairings (Weil, (optimal) ate...) are also defined via Miller functions.

---

# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a [Miller function](#) attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

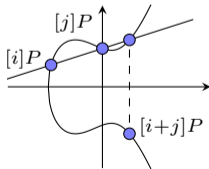
✓ Other widely used pairings (Weil, (optimal) ate...) are also defined via Miller functions.

---

[Addition law](#) on  $E \rightsquigarrow$  [addition law](#) for Miller fns  $f_{i,P}$ :

$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot (l_{[i]P,[j]P} / v_{[j]P})$$

with  $l_{R,S}$  = line through  $R$  and  $S$ , and  $v_S$  = vertical line through  $S$ .



# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a [Miller function](#) attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

✓ Other widely used pairings (Weil, (optimal) ate...) are also defined via Miller functions.

---

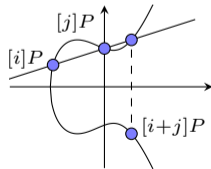
[Addition law](#) on  $E \rightsquigarrow$  [addition law](#) for Miller fns  $f_{i,P}$ :

$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot (l_{[i]P,[j]P} / v_{[j]P})$$

with  $l_{R,S}$  = line through  $R$  and  $S$ , and  $v_S$  = vertical line through  $S$ .

[Miller's algorithm](#): compute  $f_{\ell,P}(Q)$  by:

- Fix an addition chain: compute  $(P, [2]P,$
- [Alongside](#), compute  $(f_{1,P}(Q), f_{2,P}(Q),$



# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a [Miller function](#) attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

✓ Other widely used pairings (Weil, (optimal) ate...) are also defined via Miller functions.

---

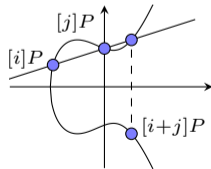
[Addition law](#) on  $E \rightsquigarrow$  [addition law](#) for Miller fns  $f_{i,P}$ :

$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot (l_{[i]P,[j]P} / v_{[j]P})$$

with  $l_{R,S}$  = line through  $R$  and  $S$ , and  $v_S$  = vertical line through  $S$ .

[Miller's algorithm](#): compute  $f_{\ell,P}(Q)$  by:

- Fix an addition chain: compute  $(P, [2]P, \dots, [i]P,$
- [Alongside](#), compute  $(f_{1,P}(Q), f_{2,P}(Q), \dots, f_{i,P}(Q),$



# How pairings are computed in practice: Miller's algorithm

Working example: the [Tate–Lichtenbaum pairing](#).

Fix degree  $\ell \in \mathbb{Z}$ , a base field  $k = \mathbb{F}_q$  containing  $\ell$ -th roots of unity  $\mu_\ell$ .

$$e_{t,\ell}: \begin{array}{ccc} G_1 \times G_2 & \rightarrow & k^\times / (k^\times)^\ell \\ (P, Q) & \mapsto & f_{\ell,P}(Q) \end{array} \quad \text{with} \quad \begin{array}{ll} G_1 & = E[\ell](k) \\ G_2 & = E(k)/[\ell]E(k) \end{array}$$

where  $f_{\ell,P} \in k(E)$  is a **Miller function** attached to  $P$ , i.e. satisfies

$$\operatorname{div} f_{\ell,P} = (\ell - 1)(0_E) + ([\ell]P) - \ell(-P) \in \operatorname{Div}^0(E)$$

✓ Other widely used pairings (Weil, (optimal) ate...) are also defined via Miller functions.

---

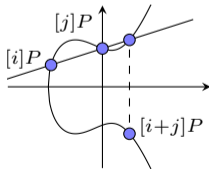
**Addition law** on  $E \rightsquigarrow$  **addition law** for Miller fns  $f_{i,P}$ :

$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot (l_{[i]P,[j]P} / v_{[j]P})$$

with  $l_{R,S}$  = line through  $R$  and  $S$ , and  $v_S$  = vertical line through  $S$ .

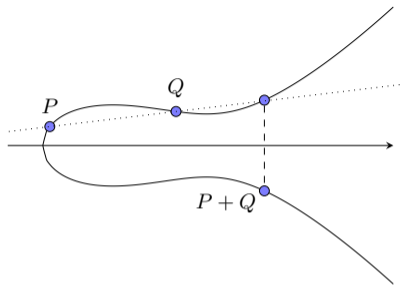
**Miller's algorithm:** compute  $f_{\ell,P}(Q)$  by:

- Fix an addition chain: compute  $(P, [2]P, \dots, [i]P, \dots, [\ell]P)$
- **Alongside**, compute  $(f_{1,P}(Q), f_{2,P}(Q), \dots, f_{i,P}(Q), \dots, f_{\ell,P}(Q))$ .



## Working with $x$ -only arithmetic

To compute line functions  $l_{R,S}$ ,  $v_R$  for Miller's algorithm, we represent points on  $E$  as  $P = (X_P : Y_P : Z_P)$ . Algebraic **group law**  $\rightsquigarrow$  tells how to add points  $P + Q$ .

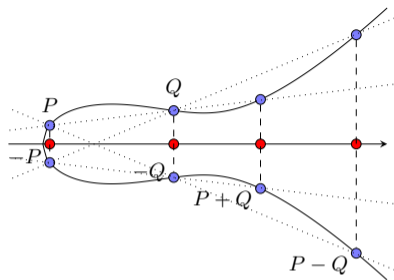


## Working with $x$ -only arithmetic

To compute line functions  $l_{R,S}$ ,  $v_R$  for Miller's algorithm, we represent points on  $E$  as  $P = (X_P : Y_P : Z_P)$ .

Algebraic **group law**  $\rightsquigarrow$  tells how to add points  $P + Q$ .

$Y_P = \pm\sqrt{g(X_P, Z_P)} \rightsquigarrow$  without  $Y$ , **sign ambiguity**:  
 $(X_P : Z_P)$  represents  $\pm P$



## Working with $x$ -only arithmetic

To compute line functions  $l_{R,S}$ ,  $v_R$  for Miller's algorithm, we represent points on  $E$  as  $P = (X_P : Y_P : Z_P)$ .

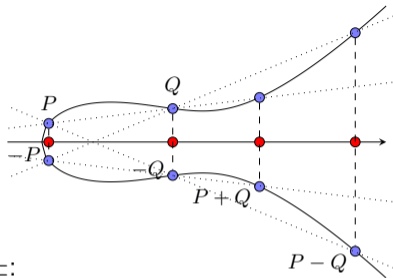
Algebraic **group law**  $\rightsquigarrow$  tells how to add points  $P + Q$ .

$Y_P = \pm\sqrt{g(X_P, Z_P)} \rightsquigarrow$  without  $Y$ , **sign ambiguity**:  
 $(X_P : Z_P)$  represents  $\pm P$

**Not a group** anymore! But there's a **pseudo-addition** on  $E/\pm$ :

xDBL:  $\pm P \mapsto \pm[2]P$ ,

xADD:  $(\pm P, \pm Q; \pm(P - Q)) \mapsto \pm(P + Q)$

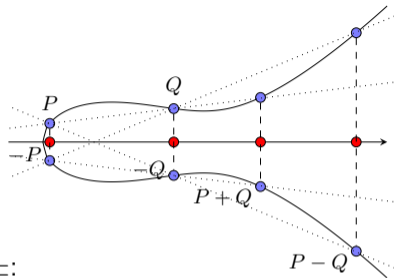


## Working with $x$ -only arithmetic

To compute line functions  $l_{R,S}$ ,  $v_R$  for Miller's algorithm, we represent points on  $E$  as  $P = (X_P : Y_P : Z_P)$ .

Algebraic **group law**  $\rightsquigarrow$  tells how to add points  $P + Q$ .

$Y_P = \pm\sqrt{g(X_P, Z_P)} \rightsquigarrow$  without  $Y$ , **sign ambiguity**:  
 $(X_P : Z_P)$  represents  $\pm P$



**Not a group** anymore! But there's a **pseudo-addition** on  $E/\pm$ :

$$\text{xDBL: } \pm P \mapsto \pm[2]P,$$

$$\text{xADD: } (\pm P, \pm Q; \pm(P - Q)) \mapsto \pm(P + Q)$$

...and it's **quite fast** to perform. **3 mult, 2 squarings** on Montgomery models  $By^2 = x^3 + Ax^2 + x$ .

$$\text{xDBL: } \begin{cases} Q = (X_P + Z_P)^2 \\ R = (X_P - Z_P)^2 \\ S = Q - R \\ [2]P = (QR : S(R + \frac{a+2}{4}S)) \end{cases}$$

$$\text{xADD: } \begin{cases} U = (X_P - Z_P)(X_Q + Z_Q) \\ V = (X_P + Z_P)(X_Q - Z_Q) \\ X_{P+Q} = Z_{P-Q} \cdot (U + V)^2 \\ Z_{P+Q} = X_{P-Q} \cdot (U - V)^2 \end{cases}$$

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$

$$[\ell]P \qquad [\ell + 1]P$$

$$\dots \qquad \dots$$

$$[2n]P \qquad [2n + 1]P$$

$$[n]P \qquad [n + 1]P$$

$$\dots \qquad \dots$$

$$P \qquad 2P$$

$$0_E \qquad P$$

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$

$0_E$

$P$

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use ***x-only*** arithmetic!

We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$

$P$	$2P$
$0_E$	$P$

# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use **x-only** arithmetic!

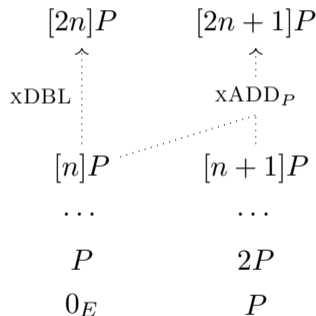
We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$



# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

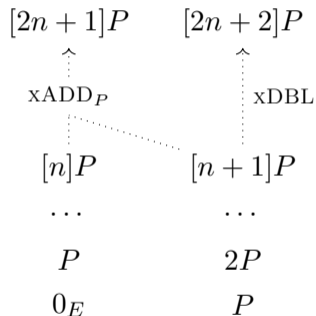
We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$



# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

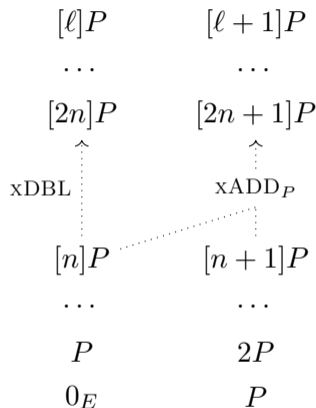
We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$



# Multiplying points by scalars: the Montgomery ladder

Goal: compute **scalar multiplication**  $P \mapsto [\ell]P$

💡  $\pm[\ell]P = [\ell](\pm P) \rightsquigarrow$  use  **$x$ -only** arithmetic!

We have operations on  $E/\pm$ :

$$\text{xDBL}: P \mapsto [2]P$$

$$\text{xADD}: (P_1, P_2; P_1 - P_2) \mapsto P_1 + P_2$$

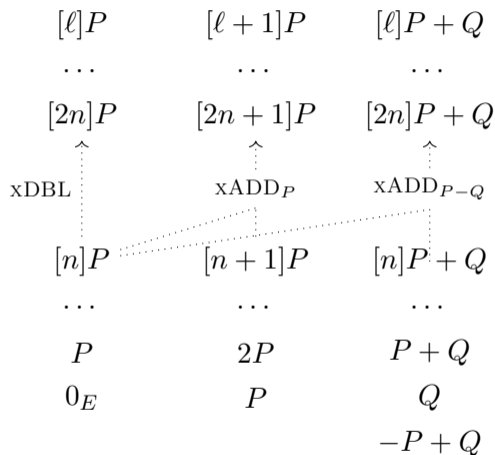
💡 **Combine** xDBL, xADD to form a

$$\text{LADDER}: (\ell, P) \mapsto ([\ell]P, [\ell + 1]P).$$

Generalization useful later:<sup>3</sup>

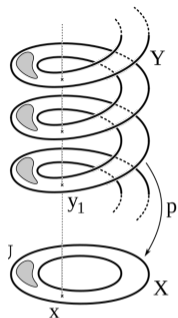
consider **3PTLADDER** with offset  $Q$ .

Needs extra input  $\pm(P - Q)$ .



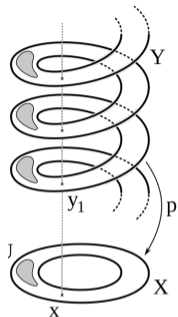
<sup>3</sup>De Feo, Jao, Plût, *Towards quantum-secure cryptosystems with isogenies*, [eprint.iacr.org/2011/506](https://eprint.iacr.org/2011/506)

## Core idea: monodromy



Walking on the helix: loop on the **projection below**  $\longleftrightarrow$  up/down one floor!

## Core idea: monodromy



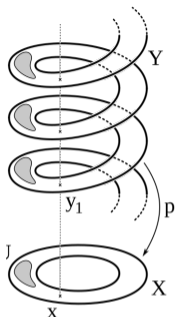
Walking on the helix: loop on the **projection below**  $\longleftrightarrow$  up/down one floor!

➡ On  $E$ : we compute  $[0]P = 0_E$ ,  $[1]P$ ,  $[2]P$ ,  $\dots$ ,  $[\ell]P = 0_E$  ...back to the start

recall:  $E \xrightarrow{\sim} \text{Pic}^0(E) = \text{Div}^0(E)/\text{Princ}(E)$

On  $\text{Pic}^0(E)$ , torsion relation  $[\ell]P = 0 \rightsquigarrow [\ell(0_E) - \ell(-P)] = 0$ .

## Core idea: monodromy



Walking on the helix: loop on the **projection below**  $\longleftrightarrow$  up/down one floor!

➡ On  $E$ : we compute  $[0]P = 0_E$ ,  $[1]P$ ,  $[2]P$ ,  $\dots$ ,  $[\ell]P = 0_E$  ...back to the start

recall:  $E \xrightarrow{\sim} \text{Pic}^0(E) = \text{Div}^0(E)/\text{Princ}(E)$

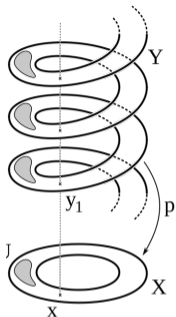
On  $\text{Pic}^0(E)$ , torsion relation  $[\ell]P = 0 \rightsquigarrow [\ell(0_E) - \ell(-P)] = 0$ .

⬆ Now look **above**: instead of its quotient, look at  $\text{Div}^0(E)$ .

$$D = \ell(0_E) - \ell(-P) = \text{div } f_{\ell, P} \neq 0 \in \text{Div}^0(E).$$

Even if  $[D] = [0]$ , the representative  $D$  carries nontrivial information: **pairings!**

## Core idea: monodromy



Walking on the helix: loop on the **projection below**  $\longleftrightarrow$  up/down one floor!

➡ On  $E$ : we compute  $[0]P = 0_E$ ,  $[1]P$ ,  $[2]P$ ,  $\dots$ ,  $[\ell]P = 0_E$  ...back to the start

recall:  $E \xrightarrow{\sim} \text{Pic}^0(E) = \text{Div}^0(E)/\text{Princ}(E)$

On  $\text{Pic}^0(E)$ , torsion relation  $[\ell]P = 0 \rightsquigarrow [\ell(0_E) - \ell(-P)] = 0$ .

⬆ Now look **above**: instead of its quotient, look at  $\text{Div}^0(E)$ .

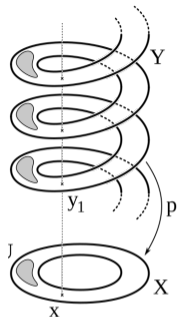
$$D = \ell(0_E) - \ell(-P) = \text{div } f_{\ell,P} \neq 0 \in \text{Div}^0(E).$$

Even if  $[D] = [0]$ , the representative  $D$  carries nontrivial information: **pairings!**



**Monodromy** in Miller's algorithm: while adding points  $0_E, P, [2]P, \dots, [\ell]P = 0_E$ , we accumulate divisor info:  $f_{0,P}(Q), \dots, f_{\ell,P}(Q) = \prod_j l_{[i_j]P, [i'_j]P}(Q) / v_{[i_j]P}(Q) = e(P, Q)$ .

## Core idea: monodromy



Walking on the helix: loop on the **projection below**  $\longleftrightarrow$  up/down one floor!

➡ On  $E$ : we compute  $[0]P = 0_E$ ,  $[1]P$ ,  $[2]P$ ,  $\dots$ ,  $[\ell]P = 0_E$  ...back to the start

recall:  $E \xrightarrow{\sim} \text{Pic}^0(E) = \text{Div}^0(E)/\text{Princ}(E)$

On  $\text{Pic}^0(E)$ , torsion relation  $[\ell]P = 0 \rightsquigarrow [\ell(0_E) - \ell(-P)] = 0$ .

⬆ Now look **above**: instead of its quotient, look at  $\text{Div}^0(E)$ .

$$D = \ell(0_E) - \ell(-P) = \text{div } f_{\ell,P} \neq 0 \in \text{Div}^0(E).$$

Even if  $[D] = [0]$ , the representative  $D$  carries nontrivial information: **pairings!**

💡 **Monodromy** in Miller's algorithm: while adding points  $0_E, P, [2]P, \dots, [\ell]P = 0_E$ , we accumulate divisor info:  $f_{0,P}(Q), \dots, f_{\ell,P}(Q) = \prod_j l_{[i_j]P, [i'_j]P}(Q) / v_{[i_j]P}(Q) = e(P, Q)$ .

💡💡 **Monodromy** already appears in the **Montgomery ladder** alone:


- Start with  $0_E = (1 : 0)$  and  $P = (X_P : Z_P)$
- Perform  $\text{LADDER}(P, \ell)$ : get  $[\ell]P = (X_{\ell P} : 0) = (1 : 0)$   
 $\rightsquigarrow X_{\ell P}$  is a monodromy factor.

➡ On  $E$ : we compute  $[0]P = 0_E, [1]P, [2]P, \dots, [\ell]P = 0_E$  ...back to the start

On  $\text{Pic}^0(E)$ , torsion relation  $[\ell]P = 0 \rightsquigarrow [\ell(0_E) - \ell(-P)] = 0$ .

$$D = \ell(0_E) - \ell(-P) = \operatorname{div} f_{\ell,P} \neq 0 \in \operatorname{Div}^0(E).$$

Even if  $[D] = [0]$ , the representative  $D$  carries nontrivial information: **pairings!**

 **Monodromy** in Miller's algorithm: while adding points  $0_E, P, [2]P, \dots, [\ell]P = 0_E$ , we accumulate divisor info:  $f_{0,P}(Q), \dots, f_{\ell,P}(Q) = \prod_j l_{[i_j]P, [i'_j]P}(Q) / v_{[i_j]P}(Q) = e(P, Q)$ .

 **Monodromy** already appears in the Montgomery ladder alone:

- Start with  $0_E = (1 : 0)$  and  $P = (X_P : Z_P)$
- Perform  $\text{LADDER}(P, \ell)$ : get  $[\ell]P = (X_{\ell P} : 0) = (1 : 0)$   
 $\rightsquigarrow X_{\ell P}$  is a monodromy factor. **PROJECTIVE COORDINATES CARRY MEANING!**

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$0_E = (1, 0) \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} [\ell]P = (X_{\ell P}, 0) \quad \text{differ by } \lambda_P = X_{\ell P}$$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$\begin{array}{ccc} 0_E = (1, 0) & & [\ell]P = (X_{\ell P}, 0) \quad \text{differ by } \lambda_P = X_{\ell P} \\ Q = (x_Q, 1) & \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} & [\ell]P + Q = (X_{\ell P+Q}, Z_{\ell P+Q}) \quad \text{differ by } \lambda_Q = Z_{\ell P+Q} \end{array}$$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$\begin{array}{ccc} 0_E = (1, 0) & & [\ell]P = (X_{\ell P}, 0) \quad \text{differ by } \lambda_P = X_{\ell P} \\ Q = (x_Q, 1) & \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} & [\ell]P + Q = (X_{\ell P+Q}, Z_{\ell P+Q}) \quad \text{differ by } \lambda_Q = Z_{\ell P+Q} \end{array}$$

From this we get the **Tate pairing**!

$$\lambda_Q / \lambda_P = e_{T, \ell}(P, Q)$$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$\begin{array}{ccc} 0_E = (1, 0) & & [\ell]P = (X_{\ell P}, 0) \quad \text{differ by } \lambda_P = X_{\ell P} \\ Q = (x_Q, 1) & \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} & [\ell]P + Q = (X_{\ell P+Q}, Z_{\ell P+Q}) \quad \text{differ by } \lambda_Q = Z_{\ell P+Q} \end{array}$$

From this we get the **Tate pairing!** squared, + garbage

$$\lambda_Q / \lambda_P = e_{T, \ell}(P, Q)^2 \cdot \text{STUFF}$$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$\begin{array}{ccc} 0_E = (1, 0) & \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} & [\ell]P = (X_{\ell P}, 0) \\ Q = (x_Q, 1) & & [\ell]P + Q = (X_{\ell P+Q}, Z_{\ell P+Q}) \end{array} \quad \begin{array}{l} \text{differ by } \lambda_P = X_{\ell P} \\ \text{differ by } \lambda_Q = Z_{\ell P+Q} \end{array}$$

From this we get the **Tate pairing!** squared, + garbage

$$\lambda_Q / \lambda_P = e_{T,\ell}(P, Q)^2 \cdot \text{STUFF}$$

$$\text{More precisely, STUFF} = \frac{(4x_P)^{\ell \cdot (\neg \ell + 1)}}{(4x_P)^{\ell \cdot \neg \ell} (4x_Q)^\ell (4x_{P-Q})^{-\ell}} \text{ depends on}^4$$

- initial input coordinates
- bit representation of  $\ell$ .

---

<sup>4</sup>notation:  $\neg \ell$  = bitwise negation of the bit representation of  $\ell$

## Montgomery ladders *almost* compute pairings

$$P = (x_P : 1) \in E[\ell], \quad Q = (x_Q : 1), \quad P - Q = (x_{P-Q} : 1)$$

We look at the 3PTLADDER where  $P, Q$  interact. Observe **monodromy factors**:

$$\begin{array}{ccc} 0_E = (1, 0) & \xrightarrow{3\text{PTLADDER}(\ell, P, Q; P-Q)} & [\ell]P = (X_{\ell P}, 0) \\ Q = (x_Q, 1) & & [\ell]P + Q = (X_{\ell P+Q}, Z_{\ell P+Q}) \end{array} \quad \begin{array}{l} \text{differ by } \lambda_P = X_{\ell P} \\ \text{differ by } \lambda_Q = Z_{\ell P+Q} \end{array}$$

From this we get the **Tate pairing!** squared, + garbage

$$\lambda_Q / \lambda_P = e_{T,\ell}(P, Q)^2 \cdot \text{STUFF}$$

$$\text{More precisely, } \text{STUFF} = \frac{(4x_P)^{\ell \cdot (\neg \ell + 1)}}{(4x_P)^{\ell \cdot \neg \ell} (4x_Q)^\ell (4x_{P-Q})^{-\ell}} \text{ depends on}^4$$

- initial input coordinates
- bit representation of  $\ell$ .

**Solution:** compute STUFF and divide it out...

**or better:** edit the LADDER to get rid of STUFF.

---

<sup>4</sup>notation:  $\neg \ell$  = bitwise negation of the bit representation of  $\ell$

# Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{aligned} U, V &= \dots \\ X_{P+Q} &= Z_{P-Q} (U + V)^2, \\ Z_{P+Q} &= X_{P-Q} (U - V)^2. \end{aligned}$$

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{aligned} U, V &= \dots \\ X_{P+Q} &= 2 \cdot Z_{P-Q} (U + V)^2, \\ Z_{P+Q} &= 2 \cdot X_{P-Q} (U - V)^2. \end{aligned}$$

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{aligned} U, V &= \dots \\ X_{P+Q} &= 3 \cdot Z_{P-Q} (U + V)^2, \\ Z_{P+Q} &= 3 \cdot X_{P-Q} (U - V)^2. \end{aligned}$$

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{aligned} U, V &= \dots \\ X_{P+Q} &= \lambda \cdot Z_{P-Q} (U + V)^2, \\ Z_{P+Q} &= \lambda \cdot X_{P-Q} (U - V)^2. \end{aligned}$$

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{array}{ll} U, V & = \dots \\ X_{P+Q} & = Z_{P-Q} (U + V)^2, \\ Z_{P+Q} & = X_{P-Q} (U - V)^2. \end{array} \quad \rightsquigarrow \quad \begin{array}{ll} U, V & = \dots \\ X_{P+Q} & = (4X_{P-Q})^{-1} \cdot (U + V)^2, \\ Z_{P+Q} & = (4Z_{P-Q})^{-1} \cdot (U - V)^2. \end{array}$$

We call this **cubical differential addition**.

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into  $\text{cADD}$ : **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{array}{ll} U, V & = \dots \\ X_{P+Q} & = Z_{P-Q} (U + V)^2, \\ Z_{P+Q} & = X_{P-Q} (U - V)^2. \end{array} \quad \rightsquigarrow \quad \begin{array}{ll} U, V & = \dots \\ X_{P+Q} & = (4X_{P-Q})^{-1} \cdot (U + V)^2, \\ Z_{P+Q} & = (4Z_{P-Q})^{-1} \cdot (U - V)^2. \end{array}$$

We call this **cubical differential addition**.

Set  $\text{CDBL} = \text{xDBL}$  and replace  $\text{cADD}$  into the ladder.

Then  $\text{cLADDER}(\ell, P, Q; P - Q) \mapsto (\ell P, \ell P + Q)$  in  $(X, Z)$ -coordinates:

$$\lambda'_Q / \lambda'_P = Z_{\ell P + Q} / X_{\ell P} = e_{T, \ell}(P, Q)^2 \quad \text{without extra STUFF!}$$

## Montgomery ladders compute pairings

Remember  $\text{xADD}(P, Q; P - Q) = (X_{P+Q}, Z_{P+Q})$ .

Modify into cADD: **different projective scaling** of the output  $(X_{P+Q}, Z_{P+Q})$

$$\begin{array}{ll} U, V &= \dots \\ X_{P+Q} &= Z_{P-Q} (U + V)^2, \\ Z_{P+Q} &= X_{P-Q} (U - V)^2. \end{array} \rightsquigarrow \begin{array}{ll} U, V &= \dots \\ X_{P+Q} &= (4X_{P-Q})^{-1} \cdot (U + V)^2, \\ Z_{P+Q} &= (4Z_{P-Q})^{-1} \cdot (U - V)^2. \end{array}$$

We call this **cubical differential addition**.

Set cDBL = xDBL and replace cADD into the ladder.

Then  $\text{cLADDER}(\ell, P, Q; P - Q) \mapsto (\ell P, \ell P + Q)$  in  $(X, Z)$ -coordinates:

$$\lambda'_Q / \lambda'_P = Z_{\ell P + Q} / X_{\ell P} = e_{T, \ell}(P, Q)^2 \quad \text{without extra STUFF!}$$

- We recover  $e_{T, \ell}$  exactly when  $\ell$  is odd ✓  $\ell$  even  $\longrightarrow$  small trick to avoid the square
- Just minor tweak needed in the conversion  $\text{xADD} \longrightarrow \text{cADD}$   
 $\rightsquigarrow$  easy **optimized, constant-time** implementation.<sup>5</sup>
- Inverses can be pre-computed and batched: **only one inversion** per pairing

---

<sup>5</sup>Rust and Sagemath libraries provided at <https://github.com/GiacomoPope/cubical-pairings>

## Other pairings

Just seen: from one Montgomery 3-point ladder with edited cADD  $\rightsquigarrow$

Non-reduced Tate pairing  $e_{T,\ell}(P, Q) = f_{\ell,P}(Q)$  from projective coordinates  $(X_{\ell P}, Z_{\ell P+Q})$ .

## Other pairings

**Just seen:** from one Montgomery 3-point ladder with edited cADD  $\rightsquigarrow$

Non-reduced Tate pairing  $e_{T,\ell}(P, Q) = f_{\ell,P}(Q)$  from projective coordinates  $(X_{\ell P}, Z_{\ell P+Q})$ .

What about other pairings? Also recoverable from ladders & some ratios!

## Other pairings

**Just seen:** from one Montgomery 3-point ladder with edited cADD  $\rightsquigarrow$

Non-reduced Tate pairing  $e_{T,\ell}(P, Q) = f_{\ell,P}(Q)$  from projective coordinates  $(X_{\ell P}, Z_{\ell P+Q})$ .

What about other pairings? Also recoverable from ladders & some ratios!

- **Weil pairing**

$$e_{W,\ell}: E[\ell] \times E[\ell] \rightarrow \mu_\ell \quad (P, Q) \mapsto f_{\ell,P}(Q)/f_{\ell,Q}(P)$$

This requires  $2 \cdot$  non-reduced Tate pairings  $\approx 2 \cdot$  cLADDER.

## Other pairings

**Just seen:** from one Montgomery 3-point ladder with edited cADD  $\rightsquigarrow$

Non-reduced Tate pairing  $e_{T,\ell}(P, Q) = f_{\ell,P}(Q)$  from projective coordinates  $(X_{\ell P}, Z_{\ell P+Q})$ .

What about other pairings? Also recoverable from ladders & some ratios!

- **Weil pairing**

$$e_{W,\ell}: E[\ell] \times E[\ell] \rightarrow \mu_\ell \quad (P, Q) \mapsto f_{\ell,P}(Q)/f_{\ell,Q}(P)$$

This requires  $2 \cdot$  non-reduced Tate pairings  $\approx 2 \cdot$  cLADDER.

- **ate pairing**

$$e_{A,\ell}: \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \mu_\ell \quad (P, Q) \mapsto f_{\lambda,P}(Q)^{\frac{q^k-1}{\ell}}$$

with  $\lambda \equiv q \pmod{\ell}$ ,  $\mathbb{G}_1 = E[\ell](\mathbb{F}_q^k)$ , and  $\mathbb{G}_2 = E[\ell] \cap \ker(\pi_q - [q])$ .

Here, monodromy between one (shorter) cLADDER and Frobenius  $\pi_q$ :

Projectively,  $\text{cLADDER}(\lambda, P, Q; P - Q) = [q]P + Q = \pi_q(P + Q)$ .

Algebra alert:

Some (high-level) theory behind the result

# Cubical arithmetic

We saw earlier:

- ladder with usual xADD  $\mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$  **STUFF**
- ladder with cADD  $\mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of xADD. **Not a coincidence!**

# Cubical arithmetic

We saw earlier:

- ladder with usual xADD  $\mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$  **STUFF**
- ladder with cADD  $\mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of xADD. **Not a coincidence!**

Algebraic statement: *if  $\Gamma(\mathcal{L}) = \langle X, Z \rangle$ , there's a **canonical isomorphism of line bundles***

$$t_{P_1}^* \mathcal{L} \otimes t_{P_2}^* \mathcal{L} \otimes t_{P_3}^* \mathcal{L} \otimes t_{P_1+P_2+P_3}^* \mathcal{L} \cong t_{P_2+P_3}^* \mathcal{L} \otimes t_{P_1+P_3}^* \mathcal{L} \otimes t_{P_1+P_2}^* \mathcal{L} \otimes \mathcal{L}$$

# Cubical arithmetic

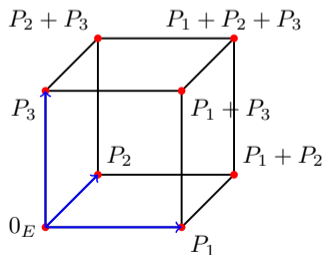
We saw earlier:

- ladder with usual  $\text{xADD} \mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2 \cdot \text{STUFF}$
- ladder with  $\text{cADD} \mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of  $\text{xADD}$ . **Not a coincidence!**

Algebraic statement: if  $\Gamma(\mathcal{L}) = \langle X, Z \rangle$ , there's a **canonical isomorphism of line bundles**

$$t_{P_1}^* \mathcal{L} \otimes t_{P_2}^* \mathcal{L} \otimes t_{P_3}^* \mathcal{L} \otimes t_{P_1+P_2+P_3}^* \mathcal{L} \cong t_{P_2+P_3}^* \mathcal{L} \otimes t_{P_1+P_3}^* \mathcal{L} \otimes t_{P_1+P_2}^* \mathcal{L} \otimes \mathcal{L}$$



# Cubical arithmetic

We saw earlier:

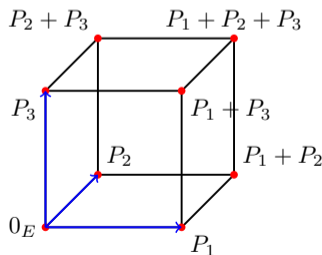
- ladder with usual  $\text{xADD} \mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2 \cdot \text{STUFF}$
- ladder with  $\text{cADD} \mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of  $\text{xADD}$ . **Not a coincidence!**

Algebraic statement: if  $\Gamma(\mathcal{L}) = \langle X, Z \rangle$ , there's a **canonical isomorphism of line bundles**

$$t_{P_1}^* \mathcal{L} \otimes t_{P_2}^* \mathcal{L} \otimes t_{P_3}^* \mathcal{L} \otimes t_{P_1+P_2+P_3}^* \mathcal{L} \cong t_{P_2+P_3}^* \mathcal{L} \otimes t_{P_1+P_3}^* \mathcal{L} \otimes t_{P_1+P_2}^* \mathcal{L} \otimes \mathcal{L}$$

Read as follows:  $t_P^* \mathcal{L} \longleftrightarrow$  scaling  $\lambda$  of coordinates  $X_P, Z_P$



# Cubical arithmetic

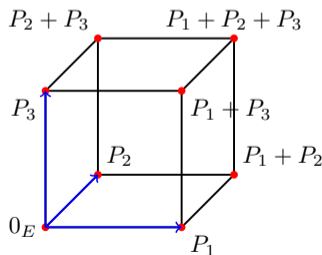
We saw earlier:

- ladder with usual  $\text{xADD} \mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2 \cdot \text{STUFF}$
- ladder with  $\text{cADD} \mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of  $\text{xADD}$ . **Not a coincidence!**

Algebraic statement: if  $\Gamma(\mathcal{L}) = \langle X, Z \rangle$ , there's a **canonical isomorphism of line bundles**

$$t_{P_1}^* \mathcal{L} \otimes t_{P_2}^* \mathcal{L} \otimes t_{P_3}^* \mathcal{L} \otimes t_{P_1+P_2+P_3}^* \mathcal{L} \cong t_{P_2+P_3}^* \mathcal{L} \otimes t_{P_1+P_3}^* \mathcal{L} \otimes t_{P_1+P_2}^* \mathcal{L} \otimes \mathcal{L}$$



Read as follows:  $t_P^* \mathcal{L} \longleftrightarrow$  scaling  $\lambda$  of coordinates  $X_P, Z_P$

Fix scaling of 7 vertices,

isomorphism above  $\implies$  canonical choice for the 8th

# Cubical arithmetic

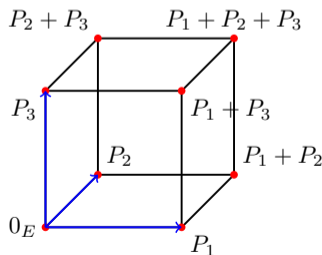
We saw earlier:

- ladder with usual xADD  $\mapsto (X_{P+Q}, Z_{P+Q}) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$  **STUFF**
- ladder with cADD  $\mapsto (X_{P+Q}/\mu, Z_{P+Q}/\mu) \rightsquigarrow Z_{\ell P+Q}/X_{\ell P} = e_{T,\ell}(P, Q)^2$

There's a **preferred** projective scaling in the output of xADD. **Not a coincidence!**

Algebraic statement: if  $\Gamma(\mathcal{L}) = \langle X, Z \rangle$ , there's a **canonical isomorphism of line bundles**

$$t_{P_1}^* \mathcal{L} \otimes t_{P_2}^* \mathcal{L} \otimes t_{P_3}^* \mathcal{L} \otimes t_{P_1+P_2+P_3}^* \mathcal{L} \cong t_{P_2+P_3}^* \mathcal{L} \otimes t_{P_1+P_3}^* \mathcal{L} \otimes t_{P_1+P_2}^* \mathcal{L} \otimes \mathcal{L}$$



Read as follows:  $t_P^* \mathcal{L} \longleftrightarrow$  scaling  $\lambda$  of coordinates  $X_P, Z_P$

Fix scaling of 7 vertices,

isomorphism above  $\implies$  canonical choice for the 8th

Then, **cADD** and cDBL are **special cases**:

Let  $(P_1, P_2, P_3) = (P, Q, -Q)$ . The vertices

$$(P, Q, -Q, P, 0, P+Q, P-Q, 0)$$

Fixing  $P, Q, P-Q$  we get  $P+Q$  uniquely!

# Cubical arithmetic as a way to get Miller functions

Main ingredient for **pairings**: compute rational fns in  $k(E)$  with **prescribed divisor**:

$$\operatorname{div} f_{\ell,P} = \ell(0_E) - \ell(-P).$$

---

# Cubical arithmetic as a way to get Miller functions

Main ingredient for pairings: compute rational fns in  $k(E)$  with prescribed divisor:

$$\operatorname{div} f_{\ell,P} = \ell(0_E) - \ell(-P).$$

---

Projective coordinates  $X, Z$  are objects living in a line bundle  $\mathcal{L}$ .

Even though they're not meromorphic functions (like  $x, y, 1$ ) in  $k(E)$ , they have a zero locus. For example,  $0_E = (1 : 0) : \rightsquigarrow Z$  has a zero at  $0_E$  (...with multiplicity 2)

# Cubical arithmetic as a way to get Miller functions

Main ingredient for **pairings**: compute rational fns in  $k(E)$  with **prescribed divisor**:

$$\operatorname{div} f_{\ell,P} = \ell(0_E) - \ell(-P).$$

---

Projective coordinates  $X, Z$  are objects living in a **line bundle**  $\mathcal{L}$ .

Even though they're not meromorphic functions (like  $x, y, 1$ ) in  $k(E)$ , they have a **zero locus**.

For example,  $0_E = (1 : 0)$ :  $\rightsquigarrow Z$  has a zero at  $0_E$  (...with multiplicity 2)

$\rightsquigarrow \exists$  reasonable notion of **divisor of zeroes**:

$$\operatorname{div}_0(Z) = 2(0_E), \quad \operatorname{div}_0(Z(\cdot + P)) = 2(-P).$$

# Cubical arithmetic as a way to get Miller functions

Main ingredient for **pairings**: compute rational fns in  $k(E)$  with **prescribed divisor**:

$$\operatorname{div} f_{\ell,P} = \ell(0_E) - \ell(-P).$$

---

Projective coordinates  $X, Z$  are objects living in a **line bundle**  $\mathcal{L}$ .

Even though they're not meromorphic functions (like  $x, y, 1$ ) in  $k(E)$ , they have a **zero locus**.

For example,  $0_E = (1 : 0)$ :  $\rightsquigarrow Z$  has a zero at  $0_E$  (...with multiplicity 2)

$\rightsquigarrow \exists$  reasonable notion of **divisor of zeroes**:

$$\operatorname{div}_0(Z) = 2(0_E), \quad \operatorname{div}_0(Z(\cdot + P)) = 2(-P).$$

**Idea**: compute some ratio  $g(\cdot) = \frac{Z(\cdot + P_1) \cdots Z(\cdot + P_m)}{Z(\cdot + Q_1) \cdots Z(\cdot + Q_m)}$ .



**Hope**:  $g \in k(E), \implies \operatorname{div} g = 2(-P_1) + \cdots + 2(-P_m) - 2(-Q_1) - \cdots - 2(-Q_m)$

**Generally not well-def**: must choose  $P_i, Q_j$  carefully, compatible with **cubical** arithmetic.

# Cubical arithmetic as a way to get Miller functions

Main ingredient for **pairings**: compute rational fns in  $k(E)$  with **prescribed divisor**:

$$\operatorname{div} f_{\ell,P} = \ell(0_E) - \ell(-P).$$

---


Projective coordinates  $X, Z$  are objects living in a **line bundle**  $\mathcal{L}$ .

Even though they're not meromorphic functions (like  $x, y, 1$ ) in  $k(E)$ , they have a **zero locus**.  
For example,  $0_E = (1 : 0) : \rightsquigarrow Z$  has a zero at  $0_E$  (...with multiplicity 2)

$\rightsquigarrow \exists$  reasonable notion of **divisor of zeroes**:

$$\operatorname{div}_0(Z) = 2(0_E), \quad \operatorname{div}_0(Z(\cdot + P)) = 2(-P).$$

**Idea**: compute some ratio  $g(\cdot) = \frac{Z(\cdot + P_1) \cdots Z(\cdot + P_m)}{Z(\cdot + Q_1) \cdots Z(\cdot + Q_m)}$ .

 **Hope**:  $g \in k(E), \implies \operatorname{div} g = 2(-P_1) + \cdots + 2(-P_m) - 2(-Q_1) - \cdots - 2(-Q_m)$

**Generally not well-def**: must choose  $P_i, Q_j$  carefully, compatible with **cubical** arithmetic.

**Miller fns**:  $P \in E[\ell]$ . Build  $g_{\ell,P} : R \mapsto \frac{Z(R + \ell P)Z(R)^{\ell-1}}{Z(P)^\ell} \rightsquigarrow \operatorname{div} g_{\ell,P} = 2 \cdot (\ell(0) - \ell(-P))$

End of the theory!

Some applications now

## Application: multi-dimensional discrete logarithms

- Consider a torsion basis  $\langle P, Q \rangle = E[N]$ , with  $N$  smooth.
- Let  $R \in E[N]$ . **DLog problem**: recover  $(a, b)$  s.t.  $R = [a]P + [b]Q$ .

## Application: multi-dimensional discrete logarithms

- Consider a torsion basis  $\langle P, Q \rangle = E[N]$ , with  $N$  smooth.
- Let  $R \in E[N]$ . **DLog problem**: recover  $(a, b)$  s.t.  $R = [a]P + [b]Q$ .

**Exploit the Weil pairing**  $e_N: E[N] \times E[N] \rightarrow \mu_N$ .

[In isogeny applications, the (2×faster) Tate pairing often shares the same properties:]

- Alternating:  $e(P, P) = 1$
- Non-degenerate: if  $P$  has order  $N$ , there is  $Q$  s.t.  $e(P, Q)$  has order  $N$ .  
 $\rightsquigarrow$  in part.,  $\langle P, Q \rangle = E[N] \iff e(P, Q) \text{ has order } N$ .

## Application: multi-dimensional discrete logarithms

- Consider a torsion basis  $\langle P, Q \rangle = E[N]$ , with  $N$  smooth.
- Let  $R \in E[N]$ . **DLog problem**: recover  $(a, b)$  s.t.  $R = [a]P + [b]Q$ .

Exploit the Weil pairing  $e_N: E[N] \times E[N] \rightarrow \mu_N$ .

[In isogeny applications, the (2×faster) Tate pairing often shares the same properties:]

- Alternating:  $e(P, P) = 1$
- Non-degenerate: if  $P$  has order  $N$ , there is  $Q$  s.t.  $e(P, Q)$  has order  $N$ .  
 $\rightsquigarrow$  in part.,  $\langle P, Q \rangle = E[N] \iff e(P, Q) \text{ has order } N$ .

Some details:

$$\zeta = e_N(P, Q) \quad \text{has order } N$$

$$h_b = e_N(R, P) = e_N([a]P + [b]Q, P) = \zeta^{-b}$$

$$h_a = e_N(R, Q) = e_N([a]P + [b]Q, Q) = \zeta^a$$

## Application: multi-dimensional discrete logarithms

- Consider a torsion basis  $\langle P, Q \rangle = E[N]$ , with  $N$  smooth.
- Let  $R \in E[N]$ . **DLog problem**: recover  $(a, b)$  s.t.  $R = [a]P + [b]Q$ .

**Exploit the Weil pairing**  $e_N: E[N] \times E[N] \rightarrow \mu_N$ .

[In isogeny applications, the (2×faster) Tate pairing often shares the same properties:]

- **Alternating**:  $e(P, P) = 1$
- **Non-degenerate**: if  $P$  has order  $N$ , there is  $Q$  s.t.  $e(P, Q)$  has order  $N$ .  
 $\rightsquigarrow$  in part.,  $\langle P, Q \rangle = E[N] \iff e(P, Q) \text{ has order } N$ .

Some details:

$$\begin{aligned}\zeta &= e_N(P, Q) \quad \text{has order } N \\ h_b &= e_N(R, P) = e_N([a]P + [b]Q, P) = \zeta^{-b} \\ h_a &= e_N(R, Q) = e_N([a]P + [b]Q, Q) = \zeta^a\end{aligned}$$

DLog in  $E[N]$   
 $\downarrow$  pairing  
DLog in  $\mu_N$ , much easier

## Application: multi-dimensional discrete logarithms

- Consider a torsion basis  $\langle P, Q \rangle = E[N]$ , with  $N$  smooth.
- Let  $R \in E[N]$ . **DLog problem**: recover  $(a, b)$  s.t.  $R = [a]P + [b]Q$ .

Exploit the Weil pairing  $e_N: E[N] \times E[N] \rightarrow \mu_N$ .

[In isogeny applications, the (2×faster) Tate pairing often shares the same properties:]

- Alternating:  $e(P, P) = 1$
- Non-degenerate: if  $P$  has order  $N$ , there is  $Q$  s.t.  $e(P, Q)$  has order  $N$ .  
 $\rightsquigarrow$  in part.,  $\langle P, Q \rangle = E[N] \iff e(P, Q) \text{ has order } N$ .

Some details:

$$\begin{aligned}\zeta &= e_N(P, Q) \quad \text{has order } N \\ h_b &= e_N(R, P) = e_N([a]P + [b]Q, P) = \zeta^{-b} \\ h_a &= e_N(R, Q) = e_N([a]P + [b]Q, Q) = \zeta^a\end{aligned}$$

DLog in  $E[N]$   
 $\downarrow$  pairing  
DLog in  $\mu_N$ , much easier

✓ Speed:  $\sim 40\%$  cost reduction w.r.t. Miller's algo. Very useful trick in isogeny protocols:  
e.g., **point compression** (SIKE  $\dagger$ , SQIsign2D):  $(a, b)$  is shorter than  $(X_R, Z_R)$ .

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
-

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .  
[alternative: trial multiplication  $P \mapsto [N/\ell_i]P$ . Pairing + order testing is much faster ✓]

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .

[alternative: trial multiplication  $P \mapsto [N/\ell_i]P$ . Pairing + order testing is much faster ✓]

Application #2: Supersingularity verification

[In CSIDH, the public key must be a supersingular curve  $E/\mathbb{F}_p \rightsquigarrow$  public key validation ✓]

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .  
[alternative: trial multiplication  $P \mapsto [N/\ell_i]P$ . Pairing + order testing is much faster  $\checkmark$ ]

Application #2: Supersingularity verification

[In CSIDH, the public key must be a supersingular curve  $E/\mathbb{F}_p \rightsquigarrow$  public key validation  $\checkmark$ ]

- Let  $E/\mathbb{F}_{p^2}$  be a supersingular curve with  $E(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1)\mathbb{Z})^2$ .
- Try to generate a  $(p+1)$ -torsion basis (#1). If SUCCESS, return “ $E$  is supersingular”.

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .  
[alternative: trial multiplication  $P \mapsto [N/\ell_i]P$ . Pairing + order testing is much faster  $\checkmark$ ]

Application #2: Supersingularity verification

[In CSIDH, the public key must be a supersingular curve  $E/\mathbb{F}_p \rightsquigarrow$  public key validation  $\checkmark$ ]

- Let  $E/\mathbb{F}_{p^2}$  be a supersingular curve with  $E(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1)\mathbb{Z})^2$ .
- Try to generate a  $(p+1)$ -torsion basis (#1). If SUCCESS, return “ $E$  is supersingular”.
- Retry few times. FAIL if we find  $P$  with  $[p+1]P \neq 0$ .  
 $\rightsquigarrow$  Probability of false negatives: 0. Probability of false positives: negligible.

## Further applications: torsion bases, supersingularity testing

Weil pairing:  $e_{W,N}: E[N] \times E[N] \rightarrow \mu_N$ .

- Non-degenerate  $\implies e(P, Q)$  has order  $N$  iff  $(P, Q)$  are a torsion basis.
- 

Use cases in CSIDH, key agreement based on group actions on isogenies.

Application #1: Torsion basis generation for very composite  $N = \prod_i \ell_i$ ,  $N \mid \#E(\mathbb{F}_q)$

- Sample random points  $P, Q$
- Do  $P, Q$  have order  $N$ ? Do they form a torsion basis? Test order of  $e(P, Q) \in \mu_N$ .  
[alternative: trial multiplication  $P \mapsto [N/\ell_i]P$ . Pairing + order testing is much faster  $\checkmark$ ]

Application #2: Supersingularity verification

[In CSIDH, the public key must be a supersingular curve  $E/\mathbb{F}_p \rightsquigarrow$  public key validation  $\checkmark$ ]

- Let  $E/\mathbb{F}_{p^2}$  be a supersingular curve with  $E(\mathbb{F}_{p^2}) \cong (\mathbb{Z}/(p+1)\mathbb{Z})^2$ .
- Try to generate a  $(p+1)$ -torsion basis (#1). If SUCCESS, return “ $E$  is supersingular”.
- Retry few times. FAIL if we find  $P$  with  $[p+1]P \neq 0$ .

$\rightsquigarrow$  Probability of false negatives: 0. Probability of false positives: negligible.

$\checkmark$  CSIDH uses even embedding degree  $k = 2 \rightsquigarrow$  only  $\sim 7\%$  cost reduction.

# Speedups in pairing-based crypto?

Main motivation of cubical pairings: **generic** pairings *in isogeny-based crypto*.  
Any benefits of the new approach on **pairing-friendly curves**?

# Speedups in pairing-based crypto?

Main motivation of cubical pairings: **generic** pairings *in isogeny-based crypto*.

Any benefits of the new approach on **pairing-friendly curves**?

↪ Parallel paper: [LRZZ25]<sup>6</sup> compares with Miller's algorithm on pairing-friendly curves.

---

<sup>6</sup>Lin, Robert, Zhao, Zheng, *Biextensions in Pairing-based Cryptography*, [eprint.iacr.org/2025/670](https://eprint.iacr.org/2025/670)

# Speedups in pairing-based crypto?

Main motivation of cubical pairings: **generic** pairings *in isogeny-based crypto*.

Any benefits of the new approach on **pairing-friendly curves**?

$\leadsto$  Parallel paper: [LRZZ25]<sup>6</sup> compares with Miller's algorithm on pairing-friendly curves.

[Def/recall **embedding degree**: say  $E$  is an ell curve over  $\mathbb{F}_q$ , and  $G \subset E[\ell](\mathbb{F}_q)$  has order  $\ell$ . The embedding degree is  $k$  if  $E[\ell]$  is only defined over  $\mathbb{F}_{q^k}$ . ]

---

<sup>6</sup>Lin, Robert, Zhao, Zheng, *Biextensions in Pairing-based Cryptography*, [eprint.iacr.org/2025/670](https://eprint.iacr.org/2025/670)

# Speedups in pairing-based crypto?

Main motivation of cubical pairings: **generic** pairings *in isogeny-based crypto*.

Any benefits of the new approach on **pairing-friendly curves**?

↪ Parallel paper: [LRZZ25]<sup>6</sup> compares with Miller's algorithm on pairing-friendly curves.

[Def/recall **embedding degree**: say  $E$  is an ell curve over  $\mathbb{F}_q$ , and  $G \subset E[\ell](\mathbb{F}_q)$  has order  $\ell$ . The embedding degree is  $k$  if  $E[\ell]$  is only defined over  $\mathbb{F}_{q^k}$ . ]

**Speedups** in **Miller** when  $k$  is even (**denominator elimination**) or composite.

✗ Both speedups **not available** in cubical arithmetic.

✓ still, **cubical arithmetic gets faster** when  $k > 1$ , i.e., some points lie in subfields  $\mathbb{F}_q \subset \mathbb{F}_{q^k}$

↪ in **some cases**, cubical arithmetic can be **faster** than Miller's algorithm:

---

<sup>6</sup>Lin, Robert, Zhao, Zheng, *Biextensions in Pairing-based Cryptography*, [eprint.iacr.org/2025/670](https://eprint.iacr.org/2025/670)

# Speedups in pairing-based crypto?

Main motivation of cubical pairings: **generic** pairings *in isogeny-based crypto*.

Any benefits of the new approach on **pairing-friendly curves**?

↪ Parallel paper: [LRZZ25]<sup>6</sup> compares with Miller's algorithm on pairing-friendly curves.

[Def/recall **embedding degree**: say  $E$  is an ell curve over  $\mathbb{F}_q$ , and  $G \subset E[\ell](\mathbb{F}_q)$  has order  $\ell$ . The embedding degree is  $k$  if  $E[\ell]$  is only defined over  $\mathbb{F}_{q^k}$ . ]

**Speedups** in **Miller** when  $k$  is even (**denominator elimination**) or composite.

✗ Both speedups **not available** in cubical arithmetic.

✓ still, **cubical arithmetic gets faster** when  $k > 1$ , i.e., some points lie in subfields  $\mathbb{F}_q \subset \mathbb{F}_{q^k}$

↪ in **some cases**, cubical arithmetic can be **faster** than Miller's algorithm:

💡 curve families with **odd prime** embedding degree  $k$  (e.g. BW13,  $k = 13$ )

---

<sup>6</sup>Lin, Robert, Zhao, Zheng, *Biextensions in Pairing-based Cryptography*, [eprint.iacr.org/2025/670](https://eprint.iacr.org/2025/670)

## Further speedups?

Main idea of the tricks we saw: replace  $xADD$  with some  $cADD$  where we change the “affine” scaling  $\lambda$  in of  $(\lambda \cdot X_{P+Q}, \lambda \cdot Z_{P+Q})$ .

## Further speedups?

Main idea of the tricks we saw: replace `xADD` with some `cADD` where we change the “affine” scaling  $\lambda$  in of  $(\lambda \cdot X_{P+Q}, \lambda \cdot Z_{P+Q})$ .

And the Montgomery ladder?

## Further speedups?

Main idea of the tricks we saw: replace  $xADD$  with some  $cADD$  where we change the “affine” scaling  $\lambda$  in of  $(\lambda \cdot X_{P+Q}, \lambda \cdot Z_{P+Q})$ .

And the Montgomery ladder?

- Good when **constant-time** is needed, code size is constrained, fast enough
- Otherwise, not the fastest way to scalar-multiply  $\ell \cdot P$

## Further speedups?

Main idea of the tricks we saw: replace  $xADD$  with some  $cADD$  where we change the “affine” scaling  $\lambda$  in of  $(\lambda \cdot X_{P+Q}, \lambda \cdot Z_{P+Q})$ .

And the Montgomery ladder?

- Good when **constant-time** is needed, code size is constrained, fast enough
- Otherwise, not the fastest way to scalar-multiply  $\ell \cdot P$

Questions:

- Can we replace it with faster **differential addition chains**?
- Or maybe double-and-add chains?
- Miller loops can be sped up by NAFs/windowing/... Can we do it too?

## Further speedups?

Main idea of the tricks we saw: replace  $xADD$  with some  $cADD$  where we change the “affine” scaling  $\lambda$  in of  $(\lambda \cdot X_{P+Q}, \lambda \cdot Z_{P+Q})$ .

And the Montgomery ladder?

- Good when **constant-time** is needed, code size is constrained, fast enough
- Otherwise, not the fastest way to scalar-multiply  $\ell \cdot P$

Questions:

- Can we replace it with faster **differential addition chains**?
- Or maybe double-and-add chains?
- Miller loops can be sped up by NAFs/windowing/... Can we do it too?

The answer in most contexts seems to be **no** :(

**Crucial** in cubical ladders: the difference points in  $xADD(P, Q; P - Q)$  are **fixed**.

- This happens in Montgomery Ladders, doesn't apply to DACs
- workarounds: use **full-coordinate**  $(X, Y, Z)$  additions  $\rightsquigarrow$  expensive.

## Recap & further directions

By modifying **projective scaling factors** in  *$x$ -only arithmetic* on elliptic curves, **Montgomery ladders** give pairings as immediate by-products.

⇒ implementation quirks: **simple**, easily constant-time, **practical**.

⇒ speedups in isogeny-based cryptography.

## Recap & further directions

By modifying **projective scaling factors** in  **$x$ -only arithmetic** on elliptic curves, **Montgomery ladders** give pairings as immediate by-products.

↪ implementation quirks: **simple**, easily constant-time, **practical**.

↪ speedups in isogeny-based cryptography.

The theory of cubical arithmetic applies much more generally:

- Other **curve models**: Theta, Weierstrass, Edwards, ...
- **Higher dimensions**: with level-2 theta models, Weil & Tate-Lichtenbaum work similarly
  - ↪ Cubical pairings already implemented in AVIsogenies (Magma), libraries in Sagemath

## Recap & further directions

By modifying **projective scaling factors** in  **$x$ -only arithmetic** on elliptic curves, **Montgomery ladders** give pairings as immediate by-products.

↪ implementation quirks: **simple**, easily constant-time, **practical**.

↪ speedups in isogeny-based cryptography.

The theory of cubical arithmetic applies much more generally:

- Other **curve models**: Theta, Weierstrass, Edwards, ...
- **Higher dimensions**: with level-2 theta models, Weil & Tate-Lichtenbaum work similarly  
↪ Cubical pairings already implemented in AVIsogenies (Magma), libraries in Sagemath

In specific contexts, alternative computations to **CLADDER** are worth comparing (e.g. **DOUBLEANDADD**, **NAFs**, ...)

## Recap & further directions

By modifying **projective scaling factors** in  **$x$ -only arithmetic** on elliptic curves, **Montgomery ladders** give pairings as immediate by-products.

↪ implementation quirks: **simple**, easily constant-time, **practical**.

↪ speedups in isogeny-based cryptography.

The theory of cubical arithmetic applies much more generally:

- Other **curve models**: Theta, Weierstrass, Edwards, ...
- **Higher dimensions**: with level-2 theta models, Weil & Tate-Lichtenbaum work similarly  
↪ Cubical pairings already implemented in AVIsogenies (Magma), libraries in Sagemath

In specific contexts, alternative computations to **C**LADDER are worth comparing (e.g. **DOUBLEANDADD**, **NAFs**, ...)

Thank you for listening! **Questions?**

## Appendix: divisors

Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

The **divisors of degree 0** on  $E$  form a group:

$$\mathrm{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

## Appendix: divisors

Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

The **divisors of degree 0** on  $E$  form a group:

$$\mathrm{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

Given a rational function  $f \in \overline{\mathbb{F}}_q(E)$ , we attach to it a **principal divisor**

$$\mathrm{div} f = \sum_{P \in E} \mathrm{ord}_P(f) \cdot (P)$$

where  $\mathrm{ord}_P(f)$  is the multiplicity of  $P$  as a zero of  $f$  if  $> 0$ , and as pole of  $f$  if  $< 0$

## Appendix: divisors

Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

The **divisors of degree 0** on  $E$  form a group:

$$\mathrm{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

Given a rational function  $f \in \overline{\mathbb{F}}_q(E)$ , we attach to it a **principal divisor**

$$\mathrm{div} f = \sum_{P \in E} \mathrm{ord}_P(f) \cdot (P)$$

where  $\mathrm{ord}_P(f)$  is the multiplicity of  $P$  as a zero of  $f$  if  $> 0$ , and as pole of  $f$  if  $< 0$

Any  $E$  elliptic curve is **isomorphic** to a quotient of  $\mathrm{Div}^0(E)$ :

$$\begin{array}{ccc} E & \xrightarrow{\sim} & \mathrm{Pic}^0(E) \\ P & \mapsto & [(P) - (0_E)] \end{array} \quad = \mathrm{Div}^0(E) / \{\text{principal divisors}\}$$

(← back to Miller's algo)

## Appendix: divisors

Let  $E/\mathbb{F}_q$  be an elliptic curve. A **divisor** on  $E$  is a formal sum

$$D = n_1 \cdot (P_1) + \dots + n_r \cdot (P_r) \quad n_i \in \mathbb{Z}, P_i \in E$$

The **divisors of degree 0** on  $E$  form a group:

$$\mathrm{Div}^0(E) = \{D = n_1(P_1) + \dots + n_r(P_r) \mid n_1 + \dots + n_r = 0\}.$$

Given a rational function  $f \in \overline{\mathbb{F}}_q(E)$ , we attach to it a **principal divisor**

$$\mathrm{div} f = \sum_{P \in E} \mathrm{ord}_P(f) \cdot (P)$$

where  $\mathrm{ord}_P(f)$  is the multiplicity of  $P$  as a zero of  $f$  if  $> 0$ , and as pole of  $f$  if  $< 0$

Any  $E$  elliptic curve is **isomorphic** to a quotient of  $\mathrm{Div}^0(E)$ :

$$\begin{array}{ccc} E & \xrightarrow{\sim} & \mathrm{Pic}^0(E) \\ P & \mapsto & [(P) - (0_E)] \end{array} = \mathrm{Div}^0(E) / \{\text{principal divisors}\}$$

(← back to Miller's algo)

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P / \lambda_Q = X_{\ell P} / Z_{\ell P + Q} = e_{T, \ell}(P, Q)^2$

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P/\lambda_Q = X_{\ell P}/Z_{\ell P+Q} = e_{T,\ell}(P, Q)^2$

The pairing has order dividing  $\ell = 2m \rightsquigarrow$  the square loses one bit of information.

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P/\lambda_Q = X_{\ell P}/Z_{\ell P+Q} = e_{T,\ell}(P, Q)^2$

The pairing has order dividing  $\ell = 2m \rightsquigarrow$  the square loses one bit of information.

**Step 1:** only compute ladder of order  $m = \ell/2$ .

$$\text{cLADDER}(m, P, Q, P - Q) \mapsto mP, mP + Q$$

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P / \lambda_Q = X_{\ell P} / Z_{\ell P + Q} = e_{T, \ell}(P, Q)^2$

The pairing has order dividing  $\ell = 2m \rightsquigarrow$  the square loses one bit of information.

**Step 1:** only compute ladder of order  $m = \ell/2$ .

$$\text{cLADDER}(m, P, Q, P - Q) \mapsto mP, mP + Q$$

**Step 2:** *Linear translations.*  $T = mP$  is a point of order 2: on the Kummer line, translation by  $T$  induces an involution. It acts linearly on coordinates, for example

$$T = (0 : 1). \quad T * (X_P, Z_P) = P + T = (Z_P, X_P)$$

$$T = (A : B) \neq (0 : 1) \quad T * (X_P, Z_P) = P + T = (AX_P - BZ_P, AZ_P - BX_P)$$

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P / \lambda_Q = X_{\ell P} / Z_{\ell P + Q} = e_{T, \ell}(P, Q)^2$

The pairing has order dividing  $\ell = 2m \rightsquigarrow$  the square loses one bit of information.

**Step 1:** only compute ladder of order  $m = \ell/2$ .

$$\text{cLADDER}(m, P, Q, P - Q) \mapsto mP, mP + Q$$

**Step 2:** *Linear translations.*  $T = mP$  is a point of order 2: on the Kummer line, translation by  $T$  induces an involution. It acts linearly on coordinates, for example

$$T = (0 : 1). \quad T * (X_P, Z_P) = P + T = (Z_P, X_P)$$

$$T = (A : B) \neq (0 : 1) \quad T * (X_P, Z_P) = P + T = (AX_P - BZ_P, AZ_P - BX_P)$$

**Step 3:** *Monodromy.*

$mP + T$ is projectively $= 0_E$	$\rightsquigarrow$ monodromy factor $\lambda'_P$
$(mP + Q) + T$ is projectively $= Q$	$\rightsquigarrow$ monodromy factor $\lambda'_Q$

## Even-degree pairings

Consider an even integer  $\ell = 2m$ .

$$P \in E[\ell](k), \quad Q \in E(k), \quad \text{cLADDER}(\ell, P, Q, P - Q) \mapsto \ell P, \ell P + Q$$

We can get the squared Tate pairing:  $\lambda_P/\lambda_Q = X_{\ell P}/Z_{\ell P+Q} = e_{T,\ell}(P, Q)^2$

The pairing has order dividing  $\ell = 2m \rightsquigarrow$  the square loses one bit of information.

**Step 1:** only compute ladder of order  $m = \ell/2$ .

$$\text{cLADDER}(m, P, Q, P - Q) \mapsto mP, mP + Q$$

**Step 2:** *Linear translations.*  $T = mP$  is a point of order 2: on the Kummer line, translation by  $T$  induces an involution. It acts linearly on coordinates, for example

$$T = (0 : 1). \quad T * (X_P, Z_P) = P + T = (Z_P, X_P)$$

$$T = (A : B) \neq (0 : 1) \quad T * (X_P, Z_P) = P + T = (AX_P - BZ_P, AZ_P - BX_P)$$

**Step 3:** *Monodromy.*  $mP + T$  is projectively  $= 0_E \rightsquigarrow$  monodromy factor  $\lambda'_P$   
 $(mP + Q) + T$  is projectively  $= Q \rightsquigarrow$  monodromy factor  $\lambda'_Q$

$$\lambda_P/\lambda_Q = X_{mP+T}/Z_{(mP+Q)+T} = e_{T,\ell}(P, Q) \quad \text{without the square!}$$

## Cubical arithmetic in different models

	cDBL	cADD
Montgomery	3M 2S	3M 2S
Theta	3M 2S	3M 3S
Weierstrass	5M 4S	8M 2S

## Appendix: Miller's algorithm

A Miller function is  $f_{\ell,P} \in k(E)$  with divisor

$$\operatorname{div} f_{\ell,P} = (\ell - 1) (0_E) + ([\ell]P) - \ell (-P) \in \operatorname{Div}^0(E)$$

These rational functions satisfy

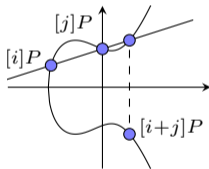
$$f_{i+j,P} = f_{i,P} \cdot f_{j,P} \cdot (l_{[i]P,[j]P} / v_{[j]P})$$

with  $l_{R,S}$  = line through  $R$  and  $S$ , and  $v_S$  = vertical line through  $S$ .

**Miller's algorithm:** compute  $f_{\ell,P}(Q)$  by:

- Fix an addition chain  $(1, 2, \dots, \ell)$
- Step by step compute  $(P, f_{1,P}(Q)), ([2]P, f_{2,P}(Q)), \dots, ([\ell]P, f_{\ell,P}(Q))$

( $\leftarrow$  back to monodromy)



## Appendix: $x$ -only Montgomery arithmetic

$$\text{xDBL: } \begin{cases} Q = (X_P + Z_P)^2 \\ R = (X_P - Z_P)^2 \\ S = Q - R \\ [2]P = (QR : S(R + \frac{a+2}{4}S)) \end{cases}$$

$$\text{xADD: } \begin{cases} U = (X_P - Z_P)(X_Q + Z_Q) \\ V = (X_P + Z_P)(X_Q - Z_Q) \\ X_{P+Q} = Z_{P-Q} \cdot (U + V)^2 \\ Z_{P+Q} = X_{P-Q} \cdot (U - V)^2 \end{cases}$$

( $\leftarrow$  go back)

## Appendix: (differential) addition chains

Fix  $\ell \in \mathbb{Z}_{>0}$  a target scalar.

An **addition chain** is a sequence of integers  $s = (n_0 = 0, n_1 = 1, n_2, n_4, \dots, n_k = \ell)$  such that

$$n \in s \implies \exists n_i, n_j \in s : n = n_i + n_j$$

Example: an addition chain for  $\ell = 9$  is  $s_9 = (0, 1, 2, 3, 5, 8, 9)$

A **differential addition chain** is a sequence of integers  $s = (n_0 = 0, n_1 = 1, n_2, n_4, \dots, n_k = \ell)$  such that

$$n \in s \implies \exists n_i, n_j \in s : n = n_i + n_j \text{ and } n_i - n_j \in s$$

Example:  $s_9$  is **not** a differential addition chain for  $\ell = 9$ :

we have  $9 = 8 + 1$ , but  $8 - 1$  is not in the sequence.

Instead this one works:  $s'_9 = (0, 1, 2, 3, 5, 7, 9)$

(← back to Miller's algo)