

Recent Advances in Parallel Implementations of Scalar Multiplication over Binary Elliptic Curves

C. Negre and J.M. Robert

RAIM 2015, Rennes,
april 8, 2015



Outline

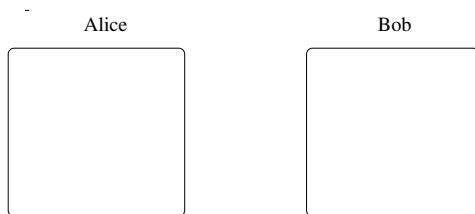
- 1 Overview of elliptic curve cryptography
- 2 Implementation of \mathbb{F}_{2^m} arithmetic
- 3 Elliptic curve arithmetic
- 4 Scalar multiplication

Outline

- 1 Overview of elliptic curve cryptography
- 2 Implementation of \mathbb{F}_{2^m} arithmetic
- 3 Elliptic curve arithmetic
- 4 Scalar multiplication

Diffie-Hellman key exchange

Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .



Diffie-Hellman key exchange

Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .

Alice

$a \leftarrow \text{random}()$

Bob

$b \leftarrow \text{random}()$

Diffie-Hellman key exchange

Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .

Alice

$a \leftarrow \text{random}()$

Computes $A = a \cdot P$

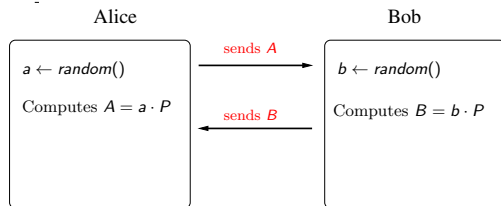
Bob

$b \leftarrow \text{random}()$

Computes $B = b \cdot P$

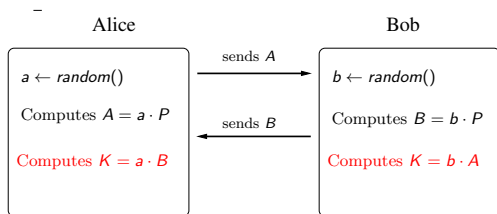
Diffie-Hellman key exchange

Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .



Diffie-Hellman key exchange

Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .

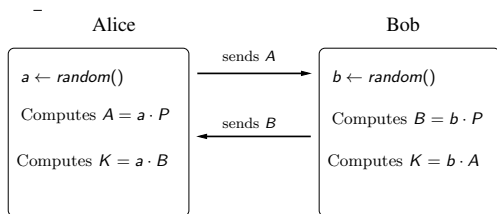


Shared secret key $K = a \cdot b \cdot P$

-

Diffie-Hellman key exchange

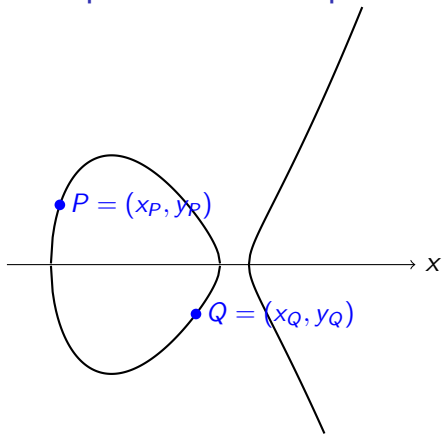
Alice and Bob agree on a group $(G, +, \mathcal{O})$ and a generating point of the group P .



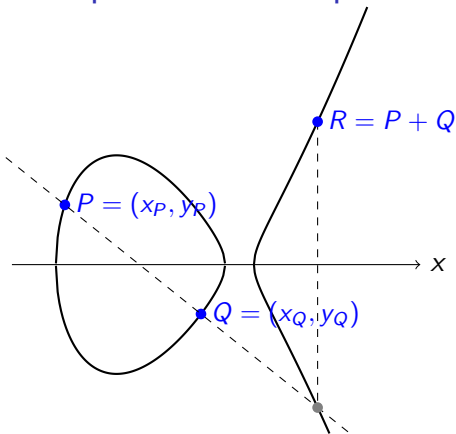
Shared secret key $K = a \cdot b \cdot P$

The main operation is the scalar multiplication $a \cdot P$.

Group law for an elliptic curve $y^2 = x^3 - 2x + 1$



Group law for an elliptic curve $y^2 = x^3 - 2x + 1$

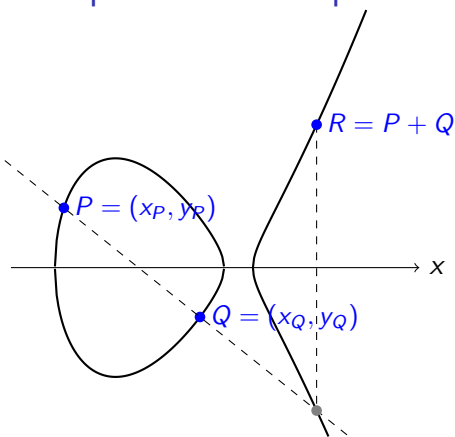


Addition (chord):

$$\begin{cases} x_R &= \lambda - x_P - x_Q \\ y_R &= y_P - \lambda(x_R - x_P) \end{cases}$$

with $\lambda = \frac{y_P - y_Q}{x_P - x_Q}$

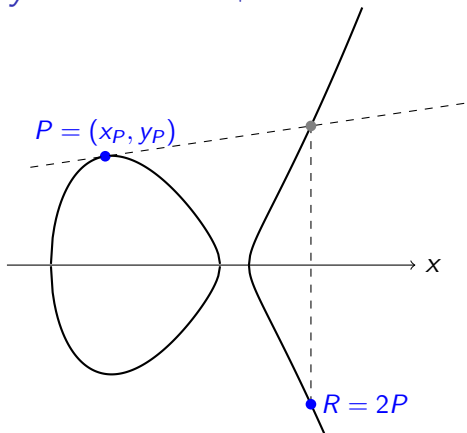
Group law for an elliptic curve $y^2 = x^3 - 2x + 1$



Addition (chord):

$$\begin{cases} x_R = \lambda - x_P - x_Q \\ y_R = y_P - \lambda(x_R - x_P) \end{cases}$$

with $\lambda = \frac{y_P - y_Q}{x_P - x_Q}$

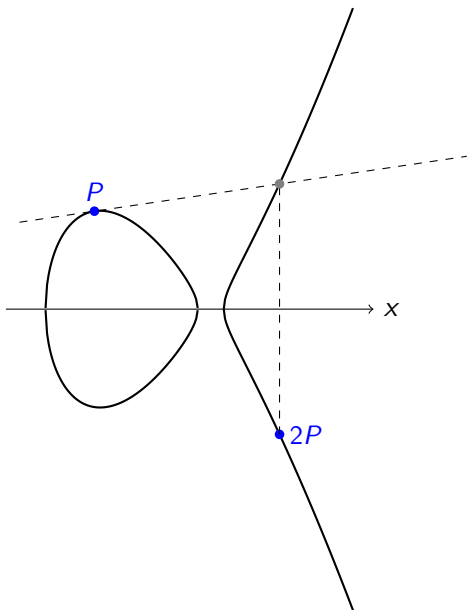


Doubling (tangent):

$$\begin{cases} x_R = \lambda - 2x_P \\ y_R = y_P - \lambda(x_R - x_P) \end{cases}$$

with $\lambda = \frac{3x_P^2 + a}{2y_P}$

Scalar multiplication : $k \cdot P$



Scalar multiplication: $7P$

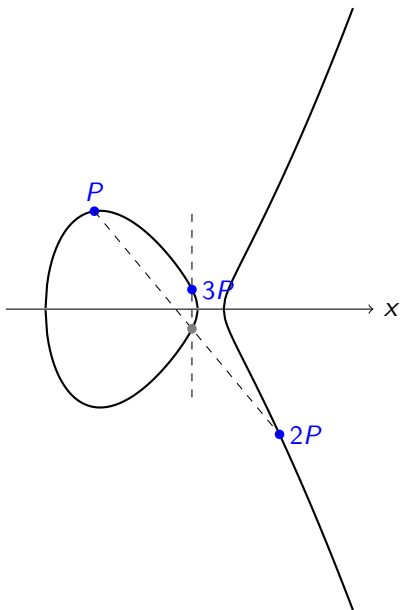
$2 \cdot P$

$$3P = (2P) + P$$

$$6P = 2 \cdot (3P)$$

$$7P = (6P) + P$$

Scalar multiplication : $k \cdot P$



Scalar multiplication: $7P$

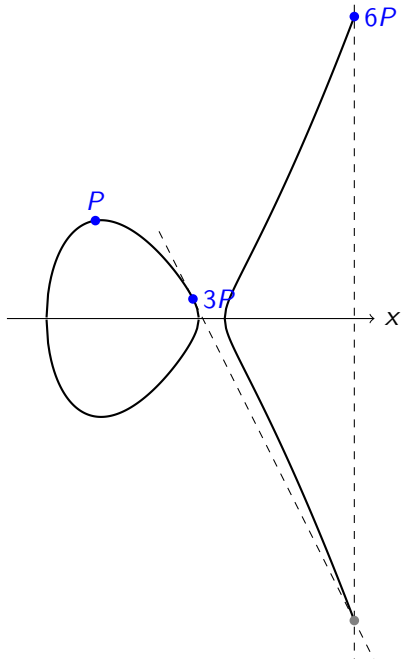
$$2 \cdot P$$

$$3P = (2P) + P$$

$$6P = 2 \cdot (3P)$$

$$7P = (6P) + P$$

Scalar multiplication : $k \cdot P$



Scalar multiplication: $7P$

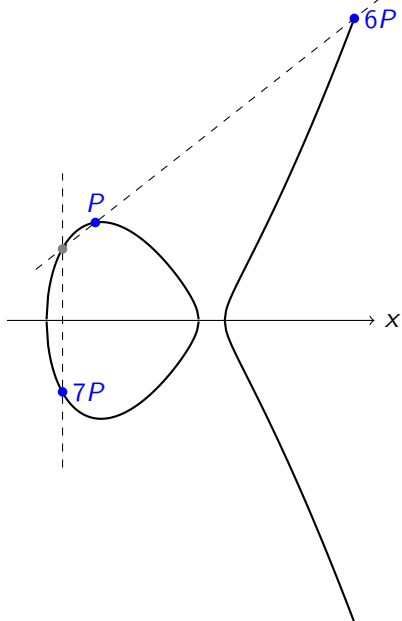
$$2 \cdot P$$

$$3P = (2P) + P$$

$$6P = 2 \cdot (3P)$$

$$7P = (6P) + P$$

Scalar multiplication : $k \cdot P$



Scalar multiplication: $7P$

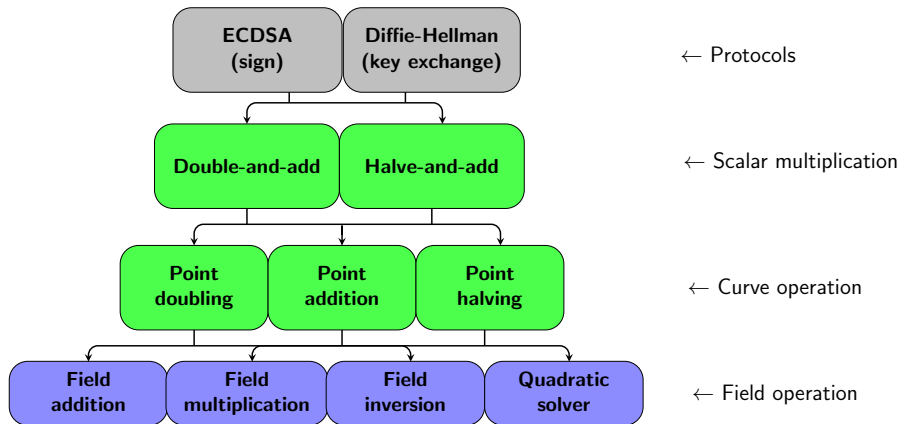
$$2 \cdot P$$

$$3P = (2P) + P$$

$$6P = 2 \cdot (3P)$$

$$7P = (6P) + P$$

Hierarchy of operations



The considered elliptic curves $E(\mathbb{F}_{2^m})$

- **Binary field:** $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$.
- **Extended binary field:** $\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(f(t))$ where $f(t)$ is irreducible.
- For $A = \sum_{i=0}^{m-1} a_i t^i$ and $B = \sum_{i=0}^{m-1} b_i t^i$ in \mathbb{F}_{2^m}

$$\text{addition : } A + B = \sum_{i=0}^{m-1} (a_i + b_i) \cdot t^i,$$

$$\text{multiplication : } A \times B = A \cdot B \pmod{f(t)}.$$

- **Binary elliptic curve:** the set of points $P = (x, y) \in \mathbb{F}_{2^m}^2$ satisfying

$$E : y^2 + xy = x^3 + ax^2 + b, \quad a, b \in \mathbb{F}_{2^m}.$$

Curve and field implemented

- **NIST curve B233**: defined over $\mathbb{F}_2[t]/(t^{233} + t^{74} + 1)$ with equation

$$E : y^2 + xy = x^3 + x^2 + b$$

where

$$b = 0x066647ede6c332c7f8c0923bb58213b333b20e9ce4281fe115f7d8f90ad, \\ N = 6901746346790563787434755862277025555839812737345013555379383634485463.$$

- **GHS curve $E(\mathbb{F}_{2^{2 \cdot 127}})$** : defined over the field $\mathbb{F}_{2^{2 \cdot 127}}$ constructed as

$$\begin{aligned} \mathbb{F}_{2^{127}} &= \mathbb{F}[t]/(t^{127} + t^{63} + 1) \\ \mathbb{F}_{2^{2 \cdot 127}} &= \mathbb{F}_{2^{127}}[u]/(u^2 + u + 1) \end{aligned}$$

with curve equation

$$E : y^2 + xy = x^3 + ux^2 + b \\ \sqrt{b} = 0xE2DA921E91E38DD1$$

and admitting an endomorphism.

Outline

- 1 Overview of elliptic curve cryptography
- 2 Implementation of \mathbb{F}_{2^m} arithmetic**
- 3 Elliptic curve arithmetic
- 4 Scalar multiplication

\mathbb{F}_{2^m} arithmetic over Intel Cores

Intel Core i3,i5 and i7 offer:

- Logical instructions XOR, AND over 128 and 256 bits.
- PCLMUL instruction computing the product of two degree 64 binary polynomials.
- PSHUFB a byte shuffling instructions .
- Shifting instruction (vector 64 bit shifts and full 128 bit shifts).

\mathbb{F}_{2^m} arithmetic over Intel Cores

Intel Core i3,i5 and i7 offer:

- Logical instructions XOR, AND over 128 and 256 bits.
- PCLMUL instruction computing the product of two degree 64 binary polynomials.
- PSHUFB a byte shuffling instructions .
- Shifting instruction (vector 64 bit shifts and full 128 bit shifts).

We will see how to implement arithmetic over $\mathbb{F}_{2^{233}}$:

- 1 Polynomial multiplication with PCLMUL.
- 2 Polynomial squaring with PSHUFB.
- 3 Reduction with shift, 128-bit XOR and AND.
- 4 Look up table for quadratic-solver.

Multiplication in $\mathbb{F}_{2^{233}}$ with Karatsuba

Karatsuba formula

For $A(x) = A_h + t^{m/2}A_l$ and $B(x) = B_h + t^{m/2}B_l$

$$A \times B = A_h B_h t^m + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) t^{m/2} + A_l B_l$$

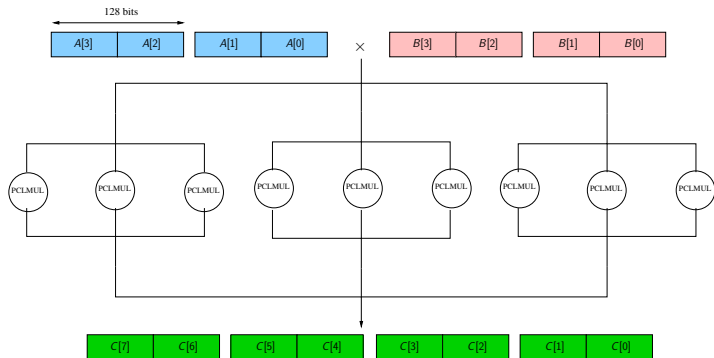
Multiplication in $\mathbb{F}_{2^{233}}$ with Karatsuba

Karatsuba formula

For $A(x) = A_h + t^{m/2}A_l$ and $B(x) = B_h + t^{m/2}B_l$

$$A \times B = A_h B_h t^m + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) t^{m/2} + A_l B_l$$

Two recursions for degree $m = 233$:



Squaring with PSHUFB

- Let a and b be two 128-bits data = 16 bytes.
- The PSHUFB instruction permute the bytes of a as specified by b

$b =$

14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

$a =$

a[15]	a[14]	a[13]	a[12]	a[11]	a[10]	a[9]	a[8]	a[7]	a[6]	a[5]	a[4]	a[3]	a[2]	a[1]	a[0]
-------	-------	-------	-------	-------	-------	------	------	------	------	------	------	------	------	------	------

PSHUFB(b , a) outputs

$c =$

a[14]	a[15]	a[12]	a[13]	a[10]	a[11]	a[8]	a[9]	a[6]	a[7]	a[4]	a[5]	a[2]	a[3]	a[0]	a[1]
-------	-------	-------	-------	-------	-------	------	------	------	------	------	------	------	------	------	------

- In other words $c[i] = a[b[i]]$

Squaring with PSHUFB

- Squaring a polynomial $b(t) = \sum_{i=0}^{m-1} b_i t^i \in \mathbb{F}_2[t]$:

$$b(t)^2 = \sum_{i=0}^{m-1} b_i t^{2i}.$$

- Aranha *et al.* 2010: Use PSHUFB for **simultaneous look-up table**:

- ▶ We store in $a[j]$ the squaring of j (seen as an element of $\mathbb{F}_2[t]$)

$$a[j] = j^2$$

for $j = 0, \dots, 16$.

- ▶ PSHUFB(b, a) computes

$$c[i] = a[b[i]] = (b[i])^2.$$

- **Squaring 128 bits = 2 PSHUFB + 1 Masking + 3 shifts.**

Square root

We express the square root of $A(t) = \sum_{i=0}^{m-1} a_i t^i$ as:

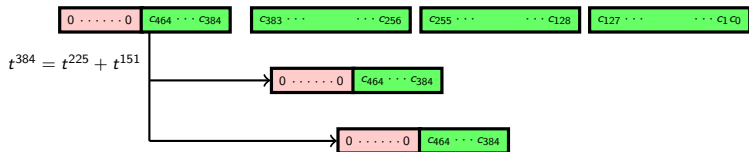
$$\begin{aligned} (A(t))^{1/2} &= \left(\overbrace{\sum_{i=0}^{\frac{m-1}{2}} a_{2i} t^{2i}}^{\text{even degree}} + \overbrace{\left(\sum_{i=0}^{\frac{m-1}{2}} a_{2i+1} t^{2i+1} \right)}^{\text{odd degree}} \right)^{1/2} \\ &= \left(\sum_{i=0}^{\frac{m-1}{2}} a_{2i} t^i \right) + \sqrt{x} \left(\sum_{i=0}^{\frac{m-1}{2}} a_{2i+1} t^i \right) \end{aligned}$$

- Masking: We separate A as A_{odd} and A_{even} .
- PSHUB: We suppress zeros in A_{odd} and A_{even} .
- Shift and XOR: we multiply A_{odd} by \sqrt{x} and XOR it to A_{even} .

Reduction modulo $f(t) = t^{233} + t^{74} + 1$



Reduction modulo $f(t) = t^{233} + t^{74} + 1$



Reduction modulo $f(t) = t^{233} + t^{74} + 1$



Reduction modulo $f(t) = t^{233} + t^{74} + 1$

$c_{383} \dots \dots c_{256}$ $c_{255} \dots \dots c_{128}$ $c_{127} \dots \dots c_1 c_0$

$0 \dots \dots 0$ $c_{464} \dots \dots c_{384}$

$0 \dots \dots 0$ $c_{464} \dots \dots c_{384}$

$c_{r,383} \dots \dots c_{r,256}$ $c_{r,255} \dots \dots c_{r,128}$ $c_{r,127} \dots \dots c_{r,1} c_{r,0}$

$t^{256} = t^{97} + t^{23}$

→ $c_{r,383} \dots \dots c_{r,256}$

→ $c_{r,383} \dots \dots c_{r,256}$

Reduction modulo $f(t) = t^{233} + t^{74} + 1$

$c_{383} \dots \dots c_{256}$ $c_{255} \dots \dots c_{128}$ $c_{127} \dots \dots c_1 c_0$

$0 \dots \dots 0$ $c_{464} \dots \dots c_{384}$

$0 \dots \dots 0$ $c_{464} \dots \dots c_{384}$

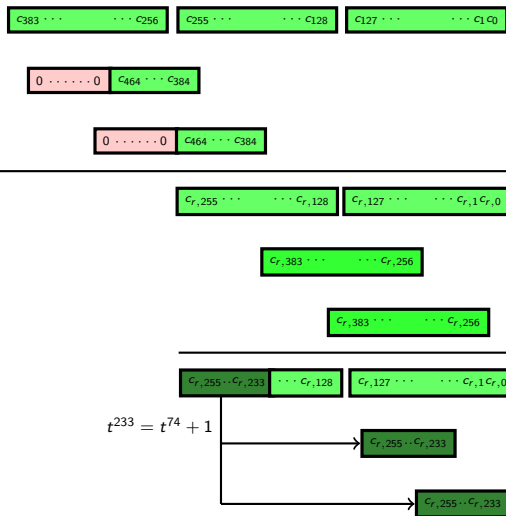
$c_{r,255} \dots \dots c_{r,128}$ $c_{r,127} \dots \dots c_{r,1} c_{r,0}$

$c_{r,383} \dots \dots c_{r,256}$

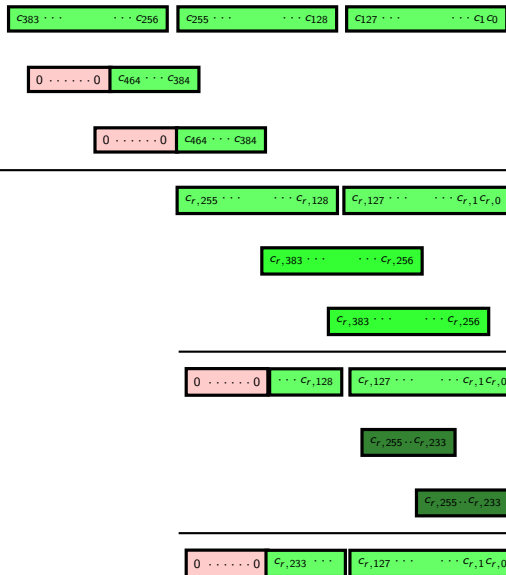
$c_{r,383} \dots \dots c_{r,256}$

$c_{r,255} \dots c_{r,233} \dots \dots c_{r,128}$ $c_{r,127} \dots \dots c_{r,1} c_{r,0}$

Reduction modulo $f(t) = t^{233} + t^{74} + 1$

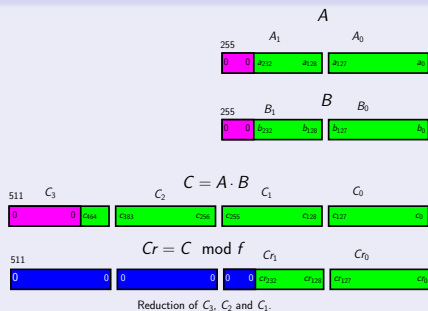


Reduction modulo $f(t) = t^{233} + t^{74} + 1$



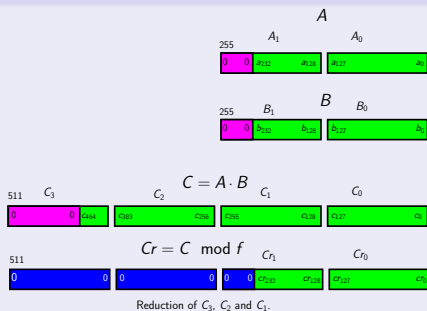
Lazy Reduction for $\mathbb{F}_{2^{233}}$

Usual reduction

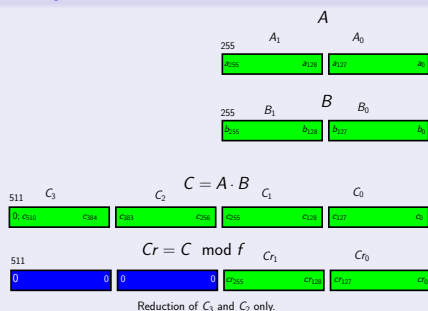


Lazy Reduction for $\mathbb{F}_{2^{233}}$

Usual reduction



Lazy reduction



Inversion

- Little Fermat theorem in \mathbb{F}_{2^m} gives

$$a^{2^m} = a \Rightarrow \left(a^{2^m-1}\right)^2 = a^{-1} \text{ if } a \neq 0.$$

Inversion

- Little Fermat theorem in \mathbb{F}_{2^m} gives

$$a^{2^m} = a \Rightarrow \left(a^{2^{m-1}-1}\right)^2 = a^{-1} \text{ if } a \neq 0.$$

- Chaining: from a^{2^u-1} and a^{2^v-1} we get $a^{2^{u+v}-1}$:

$$\left(a^{2^u-1}\right)^{2^v} \times a^{2^v-1} = a^{2^{u+v}-2^v+2^v-1} = a^{2^{u+v}-1}.$$

Inversion

- Little Fermat theorem in \mathbb{F}_{2^m} gives

$$a^{2^m} = a \Rightarrow \left(a^{2^{m-1}-1}\right)^2 = a^{-1} \text{ if } a \neq 0.$$

- Chaining: from a^{2^u-1} and a^{2^v-1} we get $a^{2^{u+v}-1}$:

$$\left(a^{2^u-1}\right)^{2^v} \times a^{2^v-1} = a^{2^{u+v}-2^v+2^v-1} = a^{2^{u+v}-1}.$$

- To invert $a \in \mathbb{F}_{2^{233}}$ we choose an addition chain to get $a^{2^{232}-1}$
 $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 14 \rightarrow 28 \rightarrow 29 \rightarrow 58 \rightarrow 116 \rightarrow 232$

$$1 \rightarrow a^{2^1-1} = a$$

$$2 = 1 + 1 \rightarrow \left(a^{2^1-1}\right)^{2^1} \times a^{2^1-1} = a^{2^2-1}$$

$$3 = 2 + 1 \rightarrow \left(a^{2^2-1}\right)^{2^1} \times a^{2^1-1} = a^{2^3-1}$$

$$6 = 3 + 3 \rightarrow \left(2^{2^3-1}\right)^{2^3} \times a^{2^3-1} = a^{2^6-1}$$

$$7 = 6 + 1 \rightarrow \left(2^{2^6-1}\right)^{2^1} \times a^{2^1-1} = a^{2^7-1}$$

$$\vdots$$

$$116 = 58 + 58 \rightarrow \left(2^{2^{58}-1}\right)^{2^{58}} \times a^{2^{58}-1} = a^{2^{116}-1}$$

$$232 = 116 + 116 \rightarrow \left(2^{2^{116}-1}\right)^{2^{116}} \times a^{2^{116}-1} = a^{2^{232}-1}$$

$$\text{and } a^{-1} = \left(a^{2^{232}-1}\right)^2$$

Quadratic solver

The solution of an equation in y

$$y^2 + y = c \text{ with } c \in \mathbb{F}_{2^m} \text{ s.t. } \text{Tr}(c) = 0 \quad (\text{where } \text{Tr}(c) = \sum_{i=0}^{m-1} c^{2^i})$$

is given by the *Half-Trace* (if m is odd):

$$y = \text{HT}(c) = \sum_{i=0}^{\frac{m-1}{2}} c^{2^{2i}}$$

Indeed, we have

$$\begin{aligned} \text{HT}(c)^2 + \text{HT}(c) &= \overbrace{\sum_{i=0}^{\frac{m-1}{2}} c^{2^{2i+1}}}^{\text{odd power}} + \overbrace{\sum_{i=0}^{\frac{m-1}{2}} c^{2^{2i}}}^{\text{even power}} \\ &= c^{2^m} + \text{Tr}(c) \end{aligned}$$

Quadratic solver: implementation

- We precompute

$$tab[i][u_3 u_2 u_1 u_0] = HT(t^{4i}(u_3 t^3 + u_2 t^2 + u_1 t + u_0))$$

for all $0 \leq i < m/4$.

- We use the linearity: $HT(c_1 + c_2) = HT(c_2) + HT(c_1)$

Quadratic solver: implementation

- We precompute

$$tab[i][u_3 u_2 u_1 u_0] = HT(t^{4i}(u_3 t^3 + u_2 t^2 + u_1 t + u_0))$$

for all $0 \leq i < m/4$.

- We use the linearity: $HT(c_1 + c_2) = HT(c_2) + HT(c_1)$

We do $m/4$ look-up table and add:

.....	$c_{11} c_{10} c_9 c_8$	$c_7 c_6 c_5 c_4$	$c_3 c_2 c_1 c_0$
-------	-------------------------	-------------------	-------------------

Quadratic solver: implementation

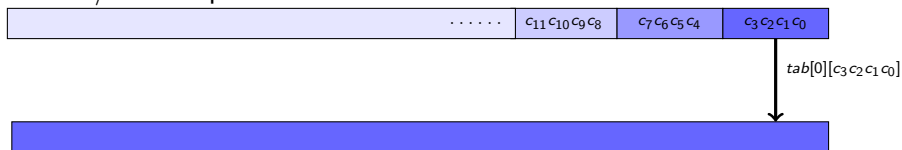
- We precompute

$$tab[i][u_3 u_2 u_1 u_0] = HT(t^{4i}(u_3 t^3 + u_2 t^2 + u_1 t + u_0))$$

for all $0 \leq i < m/4$.

- We use the linearity: $HT(c_1 + c_2) = HT(c_2) + HT(c_1)$

We do $m/4$ look-up table and add:



Quadratic solver: implementation

- We precompute

$$tab[i][u_3 u_2 u_1 u_0] = HT(t^{4i}(u_3 t^3 + u_2 t^2 + u_1 t + u_0))$$

for all $0 \leq i < m/4$.

- We use the linearity: $HT(c_1 + c_2) = HT(c_2) + HT(c_1)$

We do $m/4$ look-up table and add:



Quadratic solver: implementation

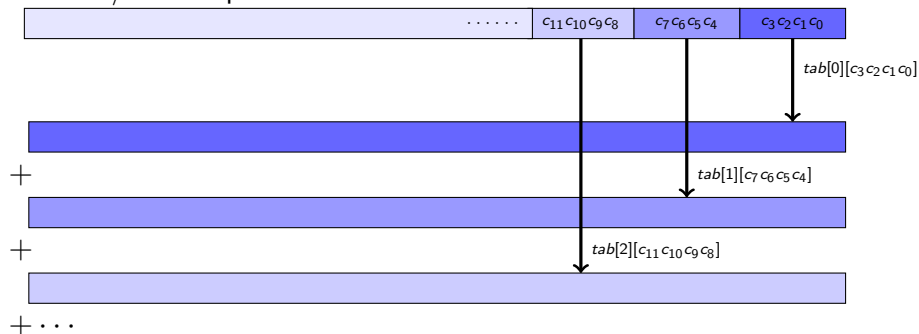
- We precompute

$$tab[i][u_3 u_2 u_1 u_0] = HT(t^{4i}(u_3 t^3 + u_2 t^2 + u_1 t + u_0))$$

for all $0 \leq i < m/4$.

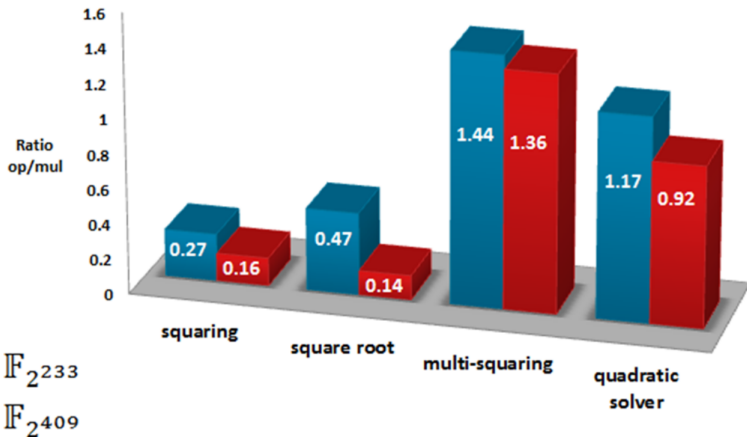
- We use the linearity: $HT(c_1 + c_2) = HT(c_2) + HT(c_1)$

We do $m/4$ look-up table and add:



Timings Core i7 SB (ratios vs multiplication)

Ratios for arithmetic operations



Outline

- 1 Overview of elliptic curve cryptography
- 2 Implementation of \mathbb{F}_{2^m} arithmetic
- 3 Elliptic curve arithmetic**
- 4 Scalar multiplication

Elliptic curves over binary fields

The formulas for point doubling and point addition are :

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \end{cases} \quad \text{with} \quad \begin{cases} \lambda = \frac{y_1 + y_2}{x_1 + x_2} \text{ if } P_1 \neq P_2 \\ \lambda = \frac{y_1}{x_1} + x_1 \text{ if } P_1 = P_2 \end{cases}$$

Elliptic curves over binary fields

The formulas for point doubling and point addition are :

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \end{cases} \quad \text{with} \quad \begin{cases} \lambda = \frac{y_1 + y_2}{x_1 + x_2} \text{ if } P_1 \neq P_2 \\ \lambda = \frac{y_1}{x_1} + x_1 \text{ if } P_1 = P_2 \end{cases}$$

- A **point doubling** requires: 1 Inv, 2 Mul, 1 Squ and 8 Add ;
- A **point addition** requires: 1 Inv, 2 Mul, 1 Squ and 9 Add;

Elliptic curves over binary fields

The formulas for point doubling and point addition are :

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \end{cases} \quad \text{with} \quad \begin{cases} \lambda = \frac{y_1 + y_2}{x_1 + x_2} \text{ if } P_1 \neq P_2 \\ \lambda = \frac{y_1}{x_1} + x_1 \text{ if } P_1 = P_2 \end{cases}$$

- A **point doubling** requires: 1 Inv, 2 Mul, 1 Squ and 8 Add ;
- A **point addition** requires: 1 Inv, 2 Mul, 1 Squ and 9 Add;

Opposite of $P = (x, y)$ is $-P = (x, x + y)$

Projective coordinates

- Lopez-Dahab projective coordinates.

$$P = (X_P : Y_P : Z_P) \text{ with } \begin{cases} x_P = \frac{X_P}{Z_P} \\ y_P = \frac{Y_P}{Z_P^2} \end{cases}$$

Projective coordinates

- Lopez-Dahab projective coordinates.

$$P = (X_P : Y_P : Z_P) \text{ with } \begin{cases} x_P = \frac{X_P}{Z_P} \\ y_P = \frac{Y_P}{Z_P^2} \end{cases}$$

- The coordinates $(X_{2P} : Y_{2P} : Z_{2P})$ of $2P$ are computed as:

$$\begin{cases} X_{2P} = X_P^4 + b \cdot Z_P^4 \\ Y_{2P} = bZ_P^4 \cdot Z_{2P} + X_{2P} \cdot (aZ_{2P} + Y_P^2 + bZ_P^4) \\ Z_{2P} = X_P^2 \cdot Z_P^2 \end{cases}$$

Projective coordinates

- Lopez-Dahab projective coordinates.

$$P = (X_P : Y_P : Z_P) \text{ with } \begin{cases} x_P = \frac{X_P}{Z_P} \\ y_P = \frac{Y_P}{Z_P^2} \end{cases}$$

- The coordinates $(X_{2P} : Y_{2P} : Z_{2P})$ of $2P$ are computed as:

$$\begin{cases} X_{2P} = X_P^4 + b \cdot Z_P^4 \\ Y_{2P} = bZ_P^4 \cdot Z_{2P} + X_{2P} \cdot (aZ_{2P} + Y_P^2 + bZ_P^4) \\ Z_{2P} = X_P^2 \cdot Z_P^2 \end{cases}$$

- Other interesting coordinates systems:
 - ▶ Kim-Kim (2006) $(X, Y, Z, T) \cong (X/Z, Y/T)$,
 - ▶ Lambda-projective coordinates (Oliveira *et al.* 2013) $(X, L, Z) \cong (X/Z, L/Z(X/Z) + (X/Z)^2)$.

Point halving

- Point doubling $Q = 2P$:

$$Q = (u, v) \quad \text{from} \quad P = (x, y)$$

$$\lambda = x + y/x, \quad (1)$$

$$u = \lambda^2 + \lambda + a, \quad (2)$$

$$v = x^2 + u(\lambda + 1). \quad (3)$$

Point halving

- Point doubling $Q = 2P$:

$$Q = (u, v) \quad \text{from} \quad P = (x, y)$$

$$\lambda = x + y/x, \quad (1)$$

$$u = \lambda^2 + \lambda + a, \quad (2)$$

$$v = x^2 + u(\lambda + 1). \quad (3)$$

- Point halving $P = \frac{1}{2}Q$: we compute x, y in terms of u, v as follows:
 - ▶ we solve (2): $\lambda^2 + \lambda = u + a \rightarrow \lambda$ (*Quadratic solver*);
 - ▶ we solve (3): $x^2 = v + u(\lambda + 1) \rightarrow x$ (*Square root*);
 - ▶ we get y with (1): $y = \lambda x + x^2$.

Point halving

- Point doubling $Q = 2P$:

$$Q = (u, v) \quad \text{from} \quad P = (x, y)$$

$$\lambda = x + y/x, \quad (1)$$

$$u = \lambda^2 + \lambda + a, \quad (2)$$

$$v = x^2 + u(\lambda + 1). \quad (3)$$

- Point halving $P = \frac{1}{2}Q$: we compute x, y in terms of u, v as follows:
 - ▶ we solve (2): $\lambda^2 + \lambda = u + a \rightarrow \lambda$ (*Quadratic solver*);
 - ▶ we solve (3): $x^2 = v + u(\lambda + 1) \rightarrow x$ (*Square root*);
 - ▶ we get y with (1): $y = \lambda x + x^2$.
- A *halving* requires:

1 *Quadratic Solver*, 1 *Square Root*, 2 *M* and 1 *S*.

Cost of point operations

	Doubling	Halving	Mixed addition	Doubling + mixed add.
Affine	$2M + 1S + 1I$	$QS + SR + S + 2M$	$2M + 1S + 1I$	$4M + 2S + 2I$
Lopez-Dahab	$4M + 4S$	-	$9M + 5S$	$13M + 10S$
Kim-Kim	$4M + 5S$	-	$8M + 5S$	$12M + 10S$
Lambda Proj.	$4M + 4S$	-	$8M + 2S$	$10M + 6S$

Outline

- 1 Overview of elliptic curve cryptography
- 2 Implementation of \mathbb{F}_{2^m} arithmetic
- 3 Elliptic curve arithmetic
- 4 Scalar multiplication**

The Scalar Multiplication Algorithm : *Double-and-add*

- Let $k = (k_{\ell-1}, \dots, k_1, k_0)_2 \in \mathbb{N}$, $P \in E(\mathbb{F}_{2^m})$

$$\begin{aligned}k \cdot P &= \left(\sum_{i=0}^{\ell-1} 2^i k_i \right) \cdot P \\&= (k_0 + 2(k_1 + 2(k_2 + 2(\dots + 2k_{\ell-1}) \dots)))) \cdot P \\&= (k_0 \cdot P + 2(k_1 \cdot P + 2(k_2 \cdot P + 2(\dots + 2(k_{\ell-2} \cdot P + 2(k_{\ell-1} \cdot P)) \dots))))\end{aligned}$$

The Scalar Multiplication Algorithm : *Double-and-add*

- Let $k = (k_{\ell-1}, \dots, k_1, k_0)_2 \in \mathbb{N}$, $P \in E(\mathbb{F}_{2^m})$

$$\begin{aligned}k \cdot P &= \left(\sum_{i=0}^{\ell-1} 2^i k_i \right) \cdot P \\&= (k_0 + 2(k_1 + 2(k_2 + 2(\dots + 2k_{\ell-1}) \dots)))) \cdot P \\&= (k_0 \cdot P + 2(k_1 \cdot P + 2(k_2 \cdot P + 2(\dots + 2(k_{\ell-2} \cdot P + 2(k_{\ell-1} \cdot P)) \dots))))\end{aligned}$$

- This can be performed with the following algorithm :

```
1:  $Q \leftarrow \mathcal{O}$ 
2: for  $i = \ell - 1$  down to 0 do
3:    $Q \leftarrow 2 \cdot Q$ 
4:   if  $k_i = 1$  then
5:      $Q \leftarrow Q + P$ 
6:   end if
7: end for
8: return  $Q$ 
```

Improvements of *Double-and-add* with *NAF* and *NAF_w*.

- *NAF* makes k sparser: let $k \in [0, 2^\ell[$ satisfying $k = 2^i - 1$, then

$$(k)_2 = \underbrace{111\dots 1}_{i \text{ times}} \text{ and we can write : } (k)_{NAF} = \underbrace{100\dots 00}_{i+1 \text{ digits}} - 1.$$

- This representation decreases the average number of non-zero digits from $\ell/2$ to $\ell/3$.

Improvements of *Double-and-add* with *NAF* and *NAF_w*.

- *NAF* makes k sparser: let $k \in [0, 2^\ell[$ satisfying $k = 2^i - 1$, then

$$(k)_2 = \underbrace{111\dots 1}_{i \text{ times}} \text{ and we can write : } (k)_{NAF} = \underbrace{100\dots 00}_{i+1 \text{ digits}} - 1.$$

- This representation decreases the average number of non-zero digits from $\ell/2$ to $\ell/3$.
- *NAF_w* further reduces the average number of non-zero digits down to $\ell/(w+1)$ using larger digits :


$$\{-2^{w-1} + 1, \dots, -5, -3, -1, 0, 1, 3, 5, \dots, 2^{w-1} - 1\}.$$

- Complexity of scalar multiplication over $E(\mathbb{F}_{2^m})$:

	nb. of doublings	nb. of additions
<i>Double-and-add</i>	ℓ	$\ell/2$
<i>NAF Double-and-add</i>	ℓ	$\ell/3$
<i>NAF_w Double-and-add</i>	ℓ	$\ell/(w+1) + 2^{w-2}$

(Double,Half)-and-add parallel scalar multiplication

Recode with NAF_w $k = \underbrace{\sum_{i=0}^{\ell} k'_i 2^i}$



Double-and-add

$R \leftarrow \mathcal{O}$

//Precomputation:

$P_i = iP$ for $i \in \{1, 3, \dots, 2^{w-1} - 1\}$

for $i = \ell$ **down to** 0 **do**

$R \leftarrow 2 \cdot R$

if $k'_i \neq 0$ **then**


$R \leftarrow R + \text{sign}(k_i)P_{|k_i|}$

endif

endfor

return(R)

(Double,Half)-and-add parallel scalar multiplication

$$\text{Recode } k = \underbrace{\sum_{i=0}^{\ell} k_i' 2^{-i}}$$


Double-and-add

```
R ← 0
//Precomputation:
Pi = iP for i ∈ {1, 3, ..., 2w-1 - 1}
for i = ℓ down to 0 do
  R ← 2 · R
  if ki' ≠ 0 then
    R ← R + sign(ki)P|ki'|
  endif
endfor
return(R)
```

Halve-and-add

```
S ← P
//Initialization:
Ri = 0 for i ∈ {1, 3, ..., 2w-1 - 1}
for i = 0 to ℓ do
  S ←  $\frac{1}{2}$  · S
  if ki' ≠ 0 then
    R|ki'| ← R|ki'| + sign(ki)S
  endif
endfor
R ←  $\sum_{i=1,3,\dots,2^w-1} i \cdot R_i$ 
return(R)
```


(Double,Half)-and-add parallel scalar multiplication

$$\text{Recode } k = \underbrace{\sum_{i=0}^{\ell-s} k'_i 2^i}_{\text{Double-and-add}} + \underbrace{\sum_{i=0}^s k''_i 2^{-i}}_{\text{Halve-and-add}}$$

Double-and-add

```
R ← 0
//Precomputation:
Pi = iP for i ∈ {1, 3, ..., 2w-1 - 1}
for i = ℓ - s down to 0 do
  R ← 2 · R
  if k'i ≠ 0 then
    R ← R + sign(ki)P|ki|
  endif
endfor
return(R)
```

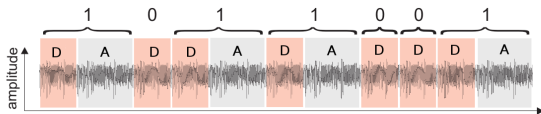
Halve-and-add

```
S ← P
//Initialization:
Ri = 0 for i ∈ {1, 3, ..., 2w-1 - 1}
for i = 0 to ℓ - s do
  S ←  $\frac{1}{2}$  · S
  if k''i ≠ 0 then
    R|k''i| ← R|k''i| + sign(ki)S
  endif
endfor
R ←  $\sum_{i=1,3,\dots,2^w-1-1} i \cdot R_i$ 
return(R)
```

Add the two points

Simple power analysis on scalar multiplication

For a scalar $k = (k_\ell, \dots, k_0)_2$ and a point $P \in E(\mathbb{F}_q)$



↑

```

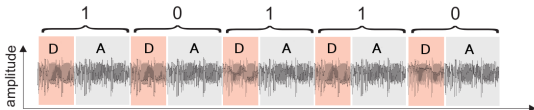
Double-and-add
R ← O
for i = ℓ - 1 to 0 do
  R ← 2 · R
  if ki = 1 then
    R ← R + P
  endif
endfor
return(R)
    
```

```

Double-and-add-always
R ← O
for i = ℓ - 1 to 0 do
  R ← 2 · R
  if ki = 1 then
    R ← R + P
  else
    R' ← R' + P
  endif
endfor
return(R)
    
```

```

Montgomery-ladder
R ← O
R' ← P
for i = ℓ - 1 to 0 do
  if ki = 1 then
    R ← R + R'
    R' ← 2 · R'
  else
    R' ← R + R'
    R ← 2 · R
  endif
endfor
return(R)
    
```



Parallelization of the Montgomery ladder (Robert 2013)

Recode and split

$$k = \underbrace{\sum_{i=0}^{\ell-s} k'_i 2^i}_{\downarrow} + \underbrace{\sum_{i=0}^s k''_i 2^{-i}}_{\downarrow}$$

Montgomery-ladder
$R \leftarrow \mathcal{O}$
$R' \leftarrow P$
for $i = \ell - s$ to 0 do
if $k'_i = 1$ then
$R \leftarrow R + R'$
$R' \leftarrow 2 \cdot R'$
else
$R' \leftarrow R + R'$
$R \leftarrow 2 \cdot R$
endif
endfor
return (R)

Montgomery-halving
$T \leftarrow \mathcal{O}$
$T' \leftarrow 2P$
for $i = s$ to 0 do
if $k''_i = 0$ then
$S \leftarrow T/2, T \leftarrow S, T' \leftarrow T' - S$
else
$S \leftarrow T'/2, T' \leftarrow S, T \leftarrow T - S$
endif
endfor
return (T)

↓
add the two results

Parallelization of the Montgomery ladder (Oliveira (2014))

Recode and split

$$k = \underbrace{\sum_{i=0}^{\ell-s} k'_i 2^i}_{\downarrow} + \underbrace{\sum_{i=0}^s k''_i 2^{-i}}_{\downarrow}$$

```
Montgomery-ladder
R ← O
R' ← P
for i = ℓ - s to 0 do
  if k'_i = 1 then
    R ← R + R'
    R' ← 2 · R'
  else
    R' ← R + R'
    R ← 2 · R
  endif
endfor
return(R)
```

```
Montgomery-halving
T ← O
T' ← 2P
precompute S_i = 1/2^i P for i = 1, ..., s
for i = 0 to s do
  if k''_i = 0 then
    T ← T + S_i
  else
    T' ← T' + S_i
  endif
endfor
return(T)
```

↓
add the two results

Parallelization with endomorphism

- Curve endomorphism (curve and group):

$$\begin{aligned}\phi: E(\mathbb{F}_q) &\rightarrow E(\mathbb{F}_q) \\ (x, y) &\mapsto (\phi_x(x, y), \phi_y(x, y))\end{aligned}$$

and there exists $\gamma_\phi \in [0, \text{ord}(P)]$ such that $\phi(P) = \gamma_\phi \cdot P$.

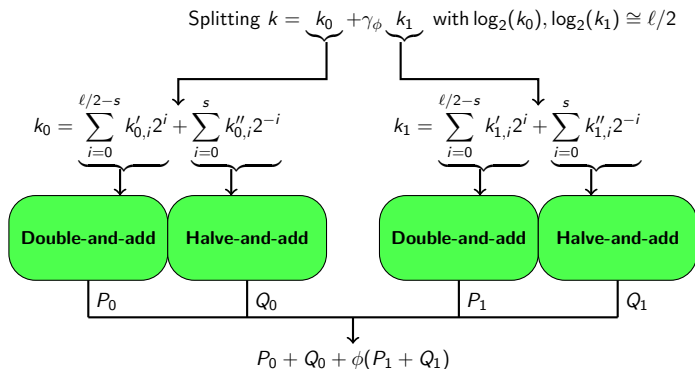
Parallelization with endomorphism

- Curve endomorphism (curve and group):

$$\begin{aligned}\phi: E(\mathbb{F}_q) &\rightarrow E(\mathbb{F}_q) \\ (x, y) &\mapsto (\phi_x(x, y), \phi_y(x, y))\end{aligned}$$

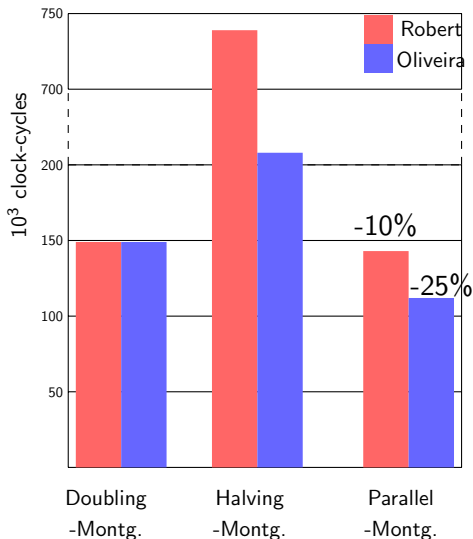
and there exists $\gamma_\phi \in [0, \text{ord}(P)]$ such that $\phi(P) = \gamma_\phi \cdot P$.

- Four thread parallelization

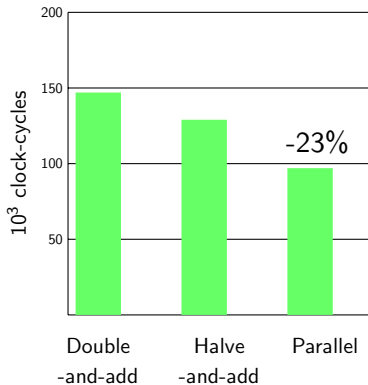


Timings: two threads (NIST B233 curve, Core i7 SB)

Montgomery-ladder



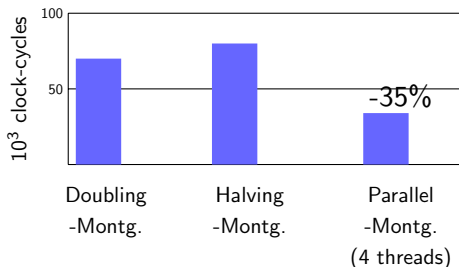
NAF_w Double, half-and-add and parallel



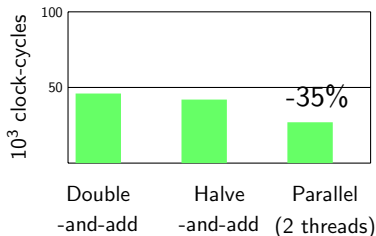
Timings: with endomorphism ($E(\mathbb{F}_{2^{2 \cdot 127}})$, Core i7 HW)

Oliveira *et al.* (2014) SAC and JCEN.

Montgomery-ladder



NAF_w Double, halve-and-add and parallel with GLV



Conclusion

- New instructions on recent Intel Cores provide significant speed-up:
 - ▶ PCLMUL,
 - ▶ PSHUFB,
 - ▶ Halving.
- New speed records:
 - ▶ A scalar multiplication over $E(\mathbb{F}_{2^m})$ requires 27 000 clock-cycles (Core i7 Haswell).
 - ▶ Better than scalar multiplication over $E(\mathbb{F}_p)$ (\cong 100 000 clock-cycles).
- For further parallelization the limitation might come from: recoding, thread management, final additions.

Thank you!

References

- **Parallel Montgomery (Oliveira)**: Thomaz Oliveira, Diego F. Aranha, Julio López Hernandez, Francisco Rodríguez-Henríquez. *Fast Point Multiplication Algorithms for Binary Elliptic Curves with and without Precomputation*. Selected Areas in Cryptography 2014: 324-344.
- **Parallel Montgomery (Robert)**: C. Negre and J.-M. Robert. *New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic*. IEEE TC, to be published.
- **Lambda Projective coordinates**: Thomaz Oliveira, Julio López, Diego F. Aranha, Francisco Rodríguez-Henríquez. *Two is the fastest prime: lambda coordinates for binary elliptic curves*. J. Cryptographic Engineering 4(1): 3-17 (2014).
- **Implementation of the squaring**: Diego F. Aranha, Julio López, Darrel Hankerson. *Efficient Software Implementation of Binary Field Arithmetic Using Vector Instruction Sets*. LATINCRYPT 2010: 144-161.