Lambda Coordinates for Binary Elliptic Curves

Thomaz Oliveira [1] Julio López [2] Diego F. Aranha [3]
Francisco Rodríguez-Henríquez [1]

[1]CINVESTAV-IPN, Mexico

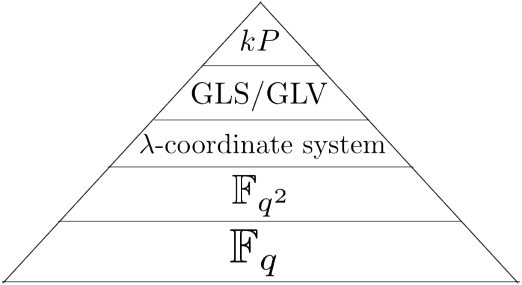
[2]University of Campinas, Brazil

[3]University of Brasília, Brazil

CHES - Santa Barbara, USA
August 23rd 2013

Outline

- Binary Field
- Elliptic Curve Arithmetic
- Scalar Multiplication
- Implementation
- Results



Binary Field

$\mathbb{F}_q$: Binary extension field of order $q = 2^m$.
Constructed by a polynomial $f(x)$ of degree m irreducible over $\mathbb{F}_2$.

$\mathbb{F}_{q^2}$: Quadratic extension of a binary field.
Constructed by a polynomial $g(u)$ of degree 2 irreducible over $\mathbb{F}_q$.

Binary Field

$\mathbb{F}_q$: Binary extension field of order $q = 2^m$.
Constructed by a polynomial $f(x)$ of degree m irreducible over $\mathbb{F}_2$.

$\mathbb{F}_{q^2}$: Quadratic extension of a binary field.
Constructed by a polynomial $g(u)$ of degree 2 irreducible over $\mathbb{F}_q$.

A careful selection of $f(x)$ and $g(u)$ is important for an efficient implementation.

Our choices: $\mathbb{F}_{2^{127}} = \mathbb{F}_2[x]/(x^{127} + x^{63} + 1)$
$\mathbb{F}_{2^{254}} = \mathbb{F}_{2^{127}}[u]/(u^2 + u + 1)$

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Given $a, b \in \mathbb{F}_q$, calculate $c = a \cdot b \mod f(x)$.

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Given $a, b \in \mathbb{F}_q$, calculate $c = a \cdot b \mod f(x)$.

The 127-bit elements in $\mathbb{F}_{2^{127}}$ can be packed into two 64-bit words.

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Given $a, b \in \mathbb{F}_q$, calculate $c = a \cdot b \mod f(x)$.

The 127-bit elements in $\mathbb{F}_{2^{127}}$ can be packed into two 64-bit words.

Polynomial multiplication can be performed using the Karatsuba method.

$$\begin{aligned} a \cdot b &= (a_1 x^{64} + a_0) \cdot (b_1 x^{64} + b_0) \\ &= (a_1 \cdot b_1) x^{128} + [(a_1 + a_0) \cdot (b_1 + b_0) + a_1 \cdot b_1 + a_0 \cdot b_0] x^{64} + a_0 \cdot b_0 \end{aligned}$$

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Given $a, b \in \mathbb{F}_q$, calculate $c = a \cdot b \mod f(x)$.

The 127-bit elements in $\mathbb{F}_{2^{127}}$ can be packed into two 64-bit words.

Polynomial multiplication can be performed using the Karatsuba method.

$$\begin{aligned} a \cdot b &= (a_1 x^{64} + a_0) \cdot (b_1 x^{64} + b_0) \\ &= (a_1 \cdot b_1) x^{128} + [(a_1 + a_0) \cdot (b_1 + b_0) + a_1 \cdot b_1 + a_0 \cdot b_0] x^{64} + a_0 \cdot b_0 \end{aligned}$$

In $\mathbb{F}_{2^{127}}$, this operation can be implemented with three carry-less multiplication instructions.

```
MUL(r1,r0,ma,mb)
t0 = _mm_xor_si128(_mm_unpacklo_epi64(ma,mb), _mm_unpackhi_epi64(ma,mb));
r0 = _mm_clmulepi64_si128(ma, mb, 0x00);
r1 = _mm_clmulepi64_si128(ma, mb, 0x11);
t0 = _mm_clmulepi64_si128(t0, t0, 0x10);
t0 = _mm_xor_si128(t0, _mm_xor_si128(r0,r1));
r0 = _mm_xor_si128(r0, _mm_slli_si128(t0, 8));
r1 = _mm_xor_si128(r1, _mm_srli_si128(t0, 8));
```

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{192+i} \equiv x^{128+i} + x^{65+i},\ i \in \{0, \ldots, 61\}$$

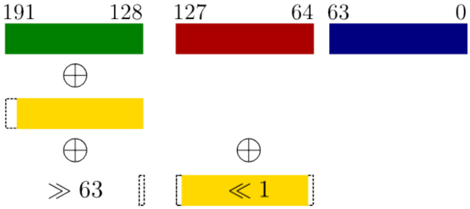
Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{192+i} \equiv x^{128+i} + x^{65+i},\ i \in \{0, \ldots, 61\}$$



Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{128+i} \equiv x^{64+i} + x^{1+i},\ i \in \{0, \ldots, 63\}$$

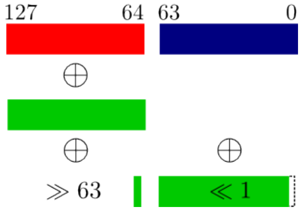
Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{128+i} \equiv x^{64+i} + x^{1+i},\ i \in \{0, \ldots, 63\}$$



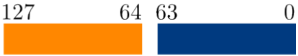
Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{127} \equiv x^{63} + 1$$



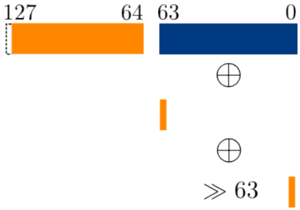
Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

$$x^{127} \equiv x^{63} + 1$$

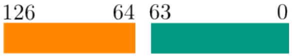


Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.



Binary Field Arithmetic

Base Field: Multiplication and Reduction

Modular reduction can be efficiently computed due to the special form of the trinomial $f(x) = x^{127} + x^{63} + 1$.

After one polynomial multiplication in $\mathbb{F}_{2^{127}}$ we have a polynomial of degree 253.

The reduction can be performed in **eleven** instructions.

```
REDUCE(t0, m1, m0)
t0 = _mm_alignr_epi8(m1,m0,8);
t0 = _mm_xor_si128(t0, m1);
m1 = _mm_slli_epi64(m1, 1);
m0 = _mm_xor_si128(m0,m1);
m1 = _mm_unpackhi_epi64(m1, t0);
m0 = _mm_xor_si128(m0,m1);
t0 = _mm_srli_epi64(t0, 63);
m0 = _mm_xor_si128(m0, t0);
m1 = _mm_unpacklo_epi64(t0, t0);
m0 = _mm_xor_si128(m0, _mm_slli_epi64(m1, 63));
```

After squaring: Taking advantage of the sparcity of the polynomial square operation, the result of this operation can be reduced using just **six** instructions.

Binary Field Arithmetic

Base Field: Other operations

Multisquaring: Performed via look-up tables of $2^4 \cdot \lceil \frac{m}{4} \rceil$ field elements.

Binary Field Arithmetic

Base Field: Other operations

Multisquaring: Performed via look-up tables of $2^4 \cdot \lceil \frac{m}{4} \rceil$ field elements.

Inversion: Can be done via the Itoh-Tsujii algorithm using the following addition chain of length 9: $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 120 \rightarrow 126$.

Binary Field Arithmetic

Base Field: Other operations

Multisquaring: Performed via look-up tables of $2^4 \cdot \lceil \frac{m}{4} \rceil$ field elements.

Inversion: Can be done via the Itoh-Tsujii algorithm using the following addition chain of length 9: $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 120 \rightarrow 126$.

Half-trace (quadratic solver): Performed via look-up tables of $2^8 \cdot \lceil \frac{m}{8} \rceil$ field elements by exploiting the linear property:
$H(c) = H(\sum_{i=0}^{m-1} c_i x^i) = \sum_{i=0}^{m-1} c_i H(x^i)$.

Binary Field Arithmetic

Quadratic extension and comparison

Taking advantage of the irreducible polynomial $g(u) = u^2 + u + 1$, all the field arithmetic in the quadratic extension $\mathbb{F}_{q^2}$ can be performed efficiently.

Binary Field Arithmetic

Quadratic extension and comparison

Taking advantage of the irreducible polynomial $g(u) = u^2 + u + 1$, all the field arithmetic in the quadratic extension $\mathbb{F}_{q^2}$ can be performed efficiently.

Multiplication:
$a \cdot b = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 \cdot b_0 + a_1 \cdot b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) + a_0 \cdot b_0)u$
with $a_0, a_1, b_0, b_1 \in \mathbb{F}_q$.

Binary Field Arithmetic

Quadratic extension and comparison

Taking advantage of the irreducible polynomial $g(u) = u^2 + u + 1$, all the field arithmetic in the quadratic extension $\mathbb{F}_{q^2}$ can be performed efficiently.

Multiplication:
$a \cdot b = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 \cdot b_0 + a_1 \cdot b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) + a_0 \cdot b_0) u$
with $a_0, a_1, b_0, b_1 \in \mathbb{F}_q$.

Squaring: $a^2 = (a_0 + a_1 u)^2 = a_0^2 + a_1^2 + a_1^2 u$.

Binary Field Arithmetic

Quadratic extension and comparison

Taking advantage of the irreducible polynomial $g(u) = u^2 + u + 1$, all the field arithmetic in the quadratic extension $\mathbb{F}_{q^2}$ can be performed efficiently.

Multiplication:
$a \cdot b = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 \cdot b_0 + a_1 \cdot b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) + a_0 \cdot b_0)u$
with $a_0, a_1, b_0, b_1 \in \mathbb{F}_q$.

Squaring: $a^2 = (a_0 + a_1 u)^2 = a_0^2 + a_1^2 + a_1^2 u$.

Inverse: $a \cdot c = (a_0 + a_1 u) \cdot (c_0 + c_1 u) = 1$. $t = a_0 \cdot a_1 + {a_0}^2 + {a_1}^2$,
$c_0 = (a_0 + a_1) \cdot t^{-1}$ and $c_1 = a_1 \cdot t^{-1}$.

Binary Field Arithmetic

Quadratic extension and comparison

Taking advantage of the irreducible polynomial $g(u) = u^2 + u + 1$, all the field arithmetic in the quadratic extension $\mathbb{F}_{q^2}$ can be performed efficiently.

Multiplication:
$a \cdot b = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 \cdot b_0 + a_1 \cdot b_1) + ((a_0 + a_1) \cdot (b_0 + b_1) + a_0 \cdot b_0)u$
with $a_0, a_1, b_0, b_1 \in \mathbb{F}_q$.

Squaring: $a^2 = (a_0 + a_1 u)^2 = a_0^2 + a_1^2 + a_1^2 u$.

Inverse: $a \cdot c = (a_0 + a_1 u) \cdot (c_0 + c_1 u) = 1$. $t = a_0 \cdot a_1 + {a_0}^2 + {a_1}^2$,
$c_0 = (a_0 + a_1) \cdot t^{-1}$ and $c_1 = a_1 \cdot t^{-1}$.

$\mathbb{F}_{q^2}$	**Multiplication**	**Square-Root**	**Squaring**	**Inversion**	**Half-Trace**
$\mathbb{F}_q$	3 mult + 4 add	2 sqrt + add	2 sqr + add	inv + 3 mult + 2 sqr + 3 add	2 ht + 2 add

Elliptic Curve Arithmetic

Binary Curves and Point Operations

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

Elliptic Curve Arithmetic

Binary Curves and Point Operations

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

The set of points $P = (x, y)$ with $x, y \in \mathbb{F}_q$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additively written abelian group with respect to the elliptic point addition operation, $E_{a,b}(\mathbb{F}_q)$.

Elliptic Curve Arithmetic

Binary Curves and Point Operations

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

The set of points $P = (x, y)$ with $x, y \in \mathbb{F}_q$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additively written abelian group with respect to the elliptic point addition operation, $E_{a,b}(\mathbb{F}_q)$.

The basic point operations:

Addition: Given $P, Q \in E_{a,b}(\mathbb{F}_q)$, with $P \neq Q$, compute $R = P + Q$.

Elliptic Curve Arithmetic

Binary Curves and Point Operations

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

The set of points $P = (x, y)$ with $x, y \in \mathbb{F}_q$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additively written abelian group with respect to the elliptic point addition operation, $E_{a,b}(\mathbb{F}_q)$.

The basic point operations:

Addition: Given $P, Q \in E_{a,b}(\mathbb{F}_q)$, with $P \neq Q$, compute $R = P + Q$.

Doubling: Given $P \in E_{a,b}(\mathbb{F}_q)$, compute $R = 2 \cdot P$.

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

The set of points $P = (x, y)$ with $x, y \in \mathbb{F}_q$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additively written abelian group with respect to the elliptic point addition operation, $E_{a,b}(\mathbb{F}_q)$.

The basic point operations:

Addition: Given $P, Q \in E_{a,b}(\mathbb{F}_q)$, with $P \neq Q$, compute $R = P + Q$.

Doubling: Given $P \in E_{a,b}(\mathbb{F}_q)$, compute $R = 2 \cdot P$.

Halving: Given $P \in E_{a,b}(\mathbb{F}_q)$, compute R such that $P = 2 \cdot R$.

Elliptic Curve Arithmetic

Binary Curves and Point Operations

Let $E/\mathbb{F}_q : y^2 + xy = x^3 + ax^2 + b$, with $a, b \in \mathbb{F}_q$ and $b \neq 0$ be a Weierstrass binary ordinary elliptic curve over $\mathbb{F}_q$.

The set of points $P = (x, y)$ with $x, y \in \mathbb{F}_q$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additively written abelian group with respect to the elliptic point addition operation, $E_{a,b}(\mathbb{F}_q)$.

The basic point operations:

Addition: Given $P, Q \in E_{a,b}(\mathbb{F}_q)$, with $P \neq Q$, compute $R = P + Q$.

Doubling: Given $P \in E_{a,b}(\mathbb{F}_q)$, compute $R = 2 \cdot P$.

Halving: Given $P \in E_{a,b}(\mathbb{F}_q)$, compute R such that $P = 2 \cdot R$.

Doubling-and-addition: Given $P, Q \in E_{a,b}(\mathbb{F}_q)$, compute R such that $R = 2 \cdot P + Q$.

Elliptic Curve Arithmetic

Lambda Projective Coordinates

λ-affine representation: Given a point $P = (x, y) \in E_{a,b}(\mathbb{F}_q)$ with $x \neq 0$, represent $P = (x, \lambda)$, where $\lambda = x + \frac{y}{x}$.

λ-affine representation: Given a point $P = (x, y) \in E_{a,b}(\mathbb{F}_q)$ with $x \neq 0$, represent $P = (x, \lambda)$, where $\lambda = x + \frac{y}{x}$.

We must have efficient formulas for addition, doubling, halving and doubling-and-addition.

λ-affine representation: Given a point $P = (x, y) \in E_{a,b}(\mathbb{F}_q)$ with $x \neq 0$, represent $P = (x, \lambda)$, where $\lambda = x + \frac{y}{x}$.

We must have efficient formulas for addition, doubling, halving and doubling-and-addition.

λ-projective point: $P = (X, L, Z)$ corresponds to the λ-affine point $(\frac{X}{Z}, \frac{L}{Z})$. The lambda-projective form of the Weierstrass equation is:

$$(L^2 + LZ + a \cdot Z^2) \cdot X^2 = X^4 + b \cdot Z^4.$$

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Doubling

Let $P = (X_P, L_P, Z_P)$ be a point in a non-supersingular curve $E_{a,b}(\mathbb{F}_q)$. Then the formula for $2P = (X_{2P}, L_{2P}, Z_{2P})$ using the λ-projective representation is given by

$$\begin{aligned} T &= L_P^2 + (L_P \cdot Z_P) + a \cdot Z_P^2 \\ X_{2P} &= T^2 \\ Z_{2P} &= T \cdot Z_P^2 \\ L_{2P} &= (X_P \cdot Z_P)^2 + X_{2P} + T \cdot (L_P \cdot Z_P) + Z_{2P}. \end{aligned}$$

Four multiplications, one multiplication by the *a*-coefficient and four squarings.

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Doubling

Let $P = (X_P, L_P, Z_P)$ be a point in a non-supersingular curve $E_{a,b}(\mathbb{F}_q)$. Then the formula for $2P = (X_{2P}, L_{2P}, Z_{2P})$ using the λ-projective representation is given by

$$\begin{aligned} T &= L_P^2 + (L_P \cdot Z_P) + a \cdot Z_P^2 \\ X_{2P} &= T^2 \\ Z_{2P} &= T \cdot Z_P^2 \\ L_{2P} &= (X_P \cdot Z_P)^2 + X_{2P} + T \cdot (L_P \cdot Z_P) + Z_{2P}. \end{aligned}$$

Four multiplications, one multiplication by the *a*-coefficient and four squarings.

If the multiplication by the b-coefficient is fast, there is an alternative formula.

$$L_{2P} = (L_P + X_P)^2 \cdot ((L_P + X_P)^2 + T + Z_P^2) + (a^2 + b) \cdot Z_P^4 + X_{2P} + (a+1) \cdot Z_{2P}.$$

Three multiplications, one multiplication by the *a*-coefficient, one multiplication by the *b*-coefficient and four squarings.

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Addition

Let $P = (X_P, L_P, Z_P)$ and $Q = (X_Q, L_Q, Z_Q)$ be points in $E_{a,b}(\mathbb{F}_q)$ with $P \neq \pm Q$. Then the addition $P + Q = (X_{P+Q}, L_{P+Q}, Z_{P+Q})$ can be computed by the formulas

$$
\begin{aligned}
A &= L_P \cdot Z_Q + L_Q \cdot Z_P \\
B &= (X_P \cdot Z_Q + X_Q \cdot Z_P)^2 \\
X_{P+Q} &= A \cdot (X_P \cdot Z_Q) \cdot (X_Q \cdot Z_P) \cdot A \\
L_{P+Q} &= (A \cdot (X_Q \cdot Z_P) + B)^2 + (A \cdot B \cdot Z_Q) \cdot (L_P + Z_P) \\
Z_{P+Q} &= (A \cdot B \cdot Z_Q) \cdot Z_P.
\end{aligned}
$$

Eleven multiplications and two squarings.

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Addition

Let $P = (X_P, L_P, Z_P)$ and $Q = (X_Q, L_Q, Z_Q)$ be points in $E_{a,b}(\mathbb{F}_q)$ with $P \neq \pm Q$. Then the addition $P + Q = (X_{P+Q}, L_{P+Q}, Z_{P+Q})$ can be computed by the formulas

$$
\begin{aligned}
A &= L_P \cdot Z_Q + L_Q \cdot Z_P \\
B &= (X_P \cdot Z_Q + X_Q \cdot Z_P)^2 \\
X_{P+Q} &= A \cdot (X_P \cdot Z_Q) \cdot (X_Q \cdot Z_P) \cdot A \\
L_{P+Q} &= (A \cdot (X_Q \cdot Z_P) + B)^2 + (A \cdot B \cdot Z_Q) \cdot (L_P + Z_P) \\
Z_{P+Q} &= (A \cdot B \cdot Z_Q) \cdot Z_P.
\end{aligned}
$$

For $Z_Q = 1$ (mixed addition),

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Addition

Let $P = (X_P, L_P, Z_P)$ and $Q = (X_Q, L_Q, Z_Q)$ be points in $E_{a,b}(\mathbb{F}_q)$ with $P \neq \pm Q$. Then the addition $P + Q = (X_{P+Q}, L_{P+Q}, Z_{P+Q})$ can be computed by the formulas

$$\begin{aligned} A &= L_P + L_Q \cdot Z_P \\ B &= (X_P + X_Q \cdot Z_P)^2 \\ X_{P+Q} &= A \cdot X_P \cdot (X_Q \cdot Z_P) \cdot A \\ L_{P+Q} &= (A \cdot (X_Q \cdot Z_P) + B)^2 + (A \cdot B) \cdot (L_P + Z_P) \\ Z_{P+Q} &= (A \cdot B) \cdot Z_P. \end{aligned}$$

Eight multiplications and two squarings.

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Doubling and Addition

Let $P = (x_P, \lambda_P)$ and $Q = (X_Q, L_Q, Z_Q)$ be points in the curve $E_{a,b}(\mathbb{F}_q)$. Then the operation $2Q + P = (X_{2Q+P}, L_{2Q+P}, Z_{2Q+P})$ can be computed as follows:

$$\begin{aligned}
T &= L_Q^2 + L_Q \cdot Z_Q + a \cdot Z_Q^2 \\
A &= X_Q^2 \cdot Z_Q^2 + T \cdot (L_Q^2 + (a + 1 + \lambda_P) \cdot Z_Q^2) \\
B &= (x_P \cdot Z_Q^2 + T)^2 \\
X_{2Q+P} &= (x_P \cdot Z_Q^2) \cdot A^2 \\
Z_{2Q+P} &= (A \cdot B \cdot Z_Q^2) \\
L_{2Q+P} &= T \cdot (A + B)^2 + (\lambda_P + 1) \cdot Z_{2Q+P}.
\end{aligned}$$

Ten multiplications, one multiplication by the *a*-constant and six squarings.

Two multiplications are saved against computing first a doubling followed by a point addition ($R = 2P$, $R = R + Q$).

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Comparison

	Coordinate systems		
	Lopez-Dahab	Lambda	
Full-addition	$13\tilde{m} + 4\tilde{s}$	$11\tilde{m} + 2\tilde{s}$	$-2\tilde{m} - 2\tilde{s}$
Mixed-addition	$8\tilde{m} + \tilde{m}_a + 5\tilde{s}$	$8\tilde{m} + 2\tilde{s}$	$-\tilde{m}_a - 3\tilde{s}$
Doubling	$3\tilde{m} + \tilde{m}_a + \tilde{m}_b + 5\tilde{s}$	$4\tilde{m} + \tilde{m}_a + 4\tilde{s}$ $3\tilde{m} + \tilde{m}_a + \tilde{m}_b + 4\tilde{s}$	$+\tilde{m} - \tilde{m}_b - \tilde{s}$ $-\tilde{s}$
Doubling and addition	$11\tilde{m} + 2\tilde{m}_a + \tilde{m}_b + 10\tilde{s}^*$	$10\tilde{m} + \tilde{m}_a + 6\tilde{s}$	$-\tilde{m} - \tilde{m}_a - \tilde{m}_b - 4\tilde{s}$

*When compared with LD doubling + mixed-addition.

Elliptic Curve Arithmetic

Lambda Projective Coordinates - Comparison

	Coordinate systems		
	Lopez-Dahab	Lambda	
Full-addition	$13\tilde{m} + 4\tilde{s}$	$11\tilde{m} + 2\tilde{s}$	$\mathbf{-2\tilde{m} - 2\tilde{s}}$
Mixed-addition	$8\tilde{m} + \tilde{m}_a + 5\tilde{s}$	$8\tilde{m} + 2\tilde{s}$	$\mathbf{-\tilde{m}_a - 3\tilde{s}}$
Doubling	$3\tilde{m} + \tilde{m}_a + \tilde{m}_b + 5\tilde{s}$	$4\tilde{m} + \tilde{m}_a + 4\tilde{s}$ $3\tilde{m}+\tilde{m}_a+\tilde{m}_b+4\tilde{s}$	$\mathbf{+\tilde{m}\ -\tilde{m}_b - \tilde{s}}$ $\mathbf{-\tilde{s}}$
Doubling and addition	$11\tilde{m}+2\tilde{m}_a+\tilde{m}_b+10\tilde{s}^*$	$10\tilde{m} + \tilde{m}_a + 6\tilde{s}$	$\mathbf{-\tilde{m} - \tilde{m}_a - \tilde{m}_b - 4\tilde{s}}$

*When compared with LD doubling + mixed-addition.

Lambda Coordinates Aftermath

More benefits and improvements derived from the lambda coordinates will be presented in the next slides.

Elliptic Curve Arithmetic

GLS Curves

The GLS curves is a large family of elliptic curves defined over $\mathbb{F}_{q^2}$ that admit efficiently computable endomorphisms. We can use the GLV method to improve significantly the point scalar multiplication by exploiting the endomorphism:

$$\psi : \tilde{E} \to \tilde{E}, \quad (x, y) \mapsto (x^{2^m}, y^{2^m} + s^{2^m} x^{2^m} + s x^{2^m}).$$

Elliptic Curve Arithmetic

GLS Curves

The GLS curves is a large family of elliptic curves defined over $\mathbb{F}_{q^2}$ that admit efficiently computable endomorphisms. We can use the GLV method to improve significantly the point scalar multiplication by exploiting the endomorphism:

$$\psi : \tilde{E} \to \tilde{E}, \quad (x, y) \mapsto (x^{2^m}, y^{2^m} + s^{2^m} x^{2^m} + s x^{2^m}).$$

For our choice of elliptic curve E defined over the quadratic field $\mathbb{F}_{q^2} \cong \mathbb{F}_{2^{127}}[u]/(u^2 + u + 1)$ we have,

$$\psi(P) = \psi(x_0 + x_1 u, y_0 + y_1 u) \mapsto ((x_0 + x_1) + x_1 u, (y_0 + y_1 + 1) + (y_1 + 1)u)$$

Elliptic Curve Arithmetic

GLS Curves

The GLS curves is a large family of elliptic curves defined over $\mathbb{F}_{q^2}$ that admit efficiently computable endomorphisms. We can use the GLV method to improve significantly the point scalar multiplication by exploiting the endomorphism:

$$\psi : \tilde{E} \to \tilde{E}, \quad (x, y) \mapsto (x^{2^m}, y^{2^m} + s^{2^m} x^{2^m} + s x^{2^m}).$$

For our choice of elliptic curve E defined over the quadratic field $\mathbb{F}_{q^2} \cong \mathbb{F}_{2^{127}}[u]/(u^2 + u + 1)$ we have,

$$\psi(P) = \psi(x_0 + x_1 u, y_0 + y_1 u) \mapsto ((x_0 + x_1) + x_1 u, (y_0 + y_1 + 1) + (y_1 + 1)u)$$

Lambda Coordinates Aftermath

For points in λ-affine representation, the endomorphism is computed as $\psi(x_0 + x_1 u, \lambda_0 + \lambda_1 u) \mapsto ((x_0 + x_1) + x_1 u, (\lambda_0 + \lambda_1) + (\lambda_1 + 1)u)$.

Scalar multiplication

Problem: Compute $Q = kP$, where $P \in E_{a,b}(\mathbb{F}_{q^2})$ is a generator of prime order r, $k \in \mathbb{Z}_r$ is a scalar of bitlength $n = |r| \approx 2m - 1$. P is not known in advance.

Scalar multiplication

Problem: Compute $Q = kP$, where $P \in E_{a,b}(\mathbb{F}_{q^2})$ is a generator of prime order r, $k \in \mathbb{Z}_r$ is a scalar of bitlength $n = |r| \approx 2m - 1$. P is not known in advance.

Methods:

- Left-to-right double-and-add:

  $Q \leftarrow \mathcal{O}$
  for i **from** $\mathbf{n-1}$ **downto** 0
      $Q \leftarrow 2Q$
      if $k_i = 1$ **then** $Q \leftarrow Q + P$

- Right-to-left halve-and-add:

  $Q \leftarrow \mathcal{O}$
  $k' \equiv 2^{n-1}k \mod r$
  for i **from** $\mathbf{n-1}$ **downto** 0
      if $k'_i = 1$ **then** $Q \leftarrow Q + P$
      $P \leftarrow P/2$

Scalar multiplication

Problem: Compute $Q = kP$, where $P \in E_{a,b}(\mathbb{F}_{q^2})$ is a generator of prime order r, $k \in \mathbb{Z}_r$ is a scalar of bitlength $n = |r| \approx 2m - 1$. P is not known in advance.

Methods:

- Left-to-right double-and-add:
  $Q \leftarrow \mathcal{O}$
  for i **from** $\mathbf{n-1}$ **downto** 0
  $\quad Q \leftarrow 2Q$
  $\quad$ **if** $k_i = 1$ **then** $Q \leftarrow Q + P$

- Right-to-left halve-and-add:
  $Q \leftarrow \mathcal{O}$
  $k' \equiv 2^{n-1}k \mod r$
  for i **from** $\mathbf{n-1}$ **downto** 0
  $\quad$ **if** $k'_i = 1$ **then** $Q \leftarrow Q + P$
  $\quad P \leftarrow P/2$

Lambda Coordinates Aftermath

Point halving function returns point P in lambda coordinates: $P = (x, \lambda)$.
Lopez-Dahab coordinate system: for the next point addition, it is necessary to return the point P to affine coordinates: $y \leftarrow (\lambda + x) \cdot x$. **Multiplication penalty.**
Lambda coordinate system: **no multiplication needed**: λ-affine coordinates are already in the input format required for the mixed-addtion function.

Scalar multiplication

Problem: Compute $Q = kP$, where $P \in E_{a,b}(\mathbb{F}_{q^2})$ is a generator of prime order r, $k \in \mathbb{Z}_r$ is a scalar of bitlength $n = |r| \approx 2m - 1$. P is not known in advance.

Methods:

- GLV
  Split the scalar k in two parts. Then $kP = k_1 P + k_2 \psi(P)$ can be performed by simultaneous multiple point techniques.

- Left-to-right double-and-add:
  $Q \leftarrow \mathcal{O}$
  $k \equiv k_1 + k_2\delta \mod r$
  for i **from** $\mathbf{n/2}$ **downto** 0
  $\quad Q \leftarrow 2Q$
  $\quad$ **if** $k_{1,i} = 1$ **then** $Q \leftarrow Q + P$
  $\quad$ **if** $k_{2,i} = 1$ **then** $Q \leftarrow Q + \psi(P)$

- Right-to-left halve-and-add:
  $Q \leftarrow \mathcal{O}$
  $k' \equiv 2^{n/2} k \mod r$
  $k' \equiv k'_1 + k'_2\delta \mod r$
  for i **from** $\mathbf{(n-1)/2}$ **downto** 0
  $\quad$ **if** $k'_{1,i} = 1$ **then** $Q \leftarrow Q + P$
  $\quad$ **if** $k'_{2,i} = 1$ **then** $Q \leftarrow Q + \psi(P)$
  $\quad P \leftarrow P/2$

Scalar multiplication

Comparison

		Double-and-add	Halve-and-add
2-GLV-GLS (LD)	pre/post sc. mult.	$1D + (2^{w-2} - 1)A + 2^{w-2}\psi$ $\frac{n}{w+1}A + \frac{n}{2}D$	$1D + (2^{w-1} - 2)A$ $\frac{n}{w+1}(A+\tilde{m}) + \frac{n}{2}H + \frac{n}{2(w+1)}\psi$

Scalar multiplication

Comparison

		Double-and-add	Halve-and-add
2-GLV-GLS (LD)	pre/post sc. mult.	$1D + (2^{w-2}-1)A + 2^{w-2}\psi$ $\frac{n}{w+1}A + \frac{n}{2}D$	$1D + (2^{w-1}-2)A$ $\frac{n}{w+1}(A+\tilde{m}) + \frac{n}{2}H + \frac{n}{2(w+1)}\psi$

Lambda Coordinates Aftermath

		Double-and-add	Halve-and-add
2-GLV-GLS (λ)	pre/post sc. mult.	$1D + (2^{w-2}-1)A + 2^{w-2}\psi$ $\frac{(2w+1)n}{2(w+1)^2}DA + \frac{w^2 n}{2(w+1)^2}D + \frac{n}{2(w+1)^2}A$	$1D + (2^{w-1}-2)A$ $\frac{n}{w+1}A + \frac{n}{2}H + \frac{n}{2(w+1)}\psi$
		−49 mult. −279 squarings *	**−51 mult. −154 squarings** *

* 4-NAF, $n = 254$, $\tilde{m}_b = \frac{2}{3}\tilde{m}$, $H = 2.48\tilde{m}$

Scalar multiplication

Parallel

Compute $k'' \equiv 2^t k \mod r$. Parameter t controls how many bits are processed by each method (double-and-add, halve-and-add) in different cores.

$$kP = \sum_{i=t}^{n-1} k_i''(2^{i-t}P) + \sum_{i=0}^{t-1} k_i'' \left(\frac{1}{2^{-(t-i)}} P\right)$$

Scalar multiplication

Parallel

Compute $k'' \equiv 2^t k \mod r$. Parameter t controls how many bits are processed by each method (double-and-add, halve-and-add) in different cores.

$$kP = \sum_{i=t}^{n-1} k_i''(2^{i-t}P) + \sum_{i=0}^{t-1} k_i'' \left(\frac{1}{2^{-(t-i)}}P\right)$$

Also, the GLV method can be combined with the parallel technique, which implies that the loop length in each core reduces to $\approx n/4$.

Scalar multiplication

Parallel

Compute $k'' \equiv 2^t k \mod r$. Parameter t controls how many bits are processed by each method (double-and-add, halve-and-add) in different cores.

$$kP = \sum_{i=t}^{n-1} k''_i (2^{i-t} P) + \sum_{i=0}^{t-1} k''_i \left(\frac{1}{2^{-(t-i)}} P \right)$$

Also, the GLV method can be combined with the parallel technique, which implies that the loop length in each core reduces to $\approx n/4$.

Algorithm 3 Parallel scalar multiplication with GLV method

Require: $P \in E(\mathbb{F}_{2^{2m}})$, scalars k_1, k_2 of bitlength $d \approx n/2$, w, constant t
Ensure: $Q = kP$

$Q \leftarrow \mathcal{O}$
for $i = d$ **downto** t **do**
  $Q \leftarrow 2Q$
  if $k_{1,i} = 1$ **then** $Q \leftarrow Q + P$
  if $k_{2,i} = 1$ **then** $Q \leftarrow Q + \psi(P)$
end for
$\{$Barrier$\}$

Initialize $Q_0 \leftarrow \mathcal{O}$
for $i = t - 1$ **downto** 0 **do**
  $P \leftarrow P/2$
  if $k_{1,i} = 1$ **then** $Q_0 \leftarrow Q_0 + P$
  if $k_{2,i} = 1$ **then** $Q_0 \leftarrow Q_0 + \psi(P)$
end for
$\{$Barrier$\}$

return $Q \leftarrow Q + Q_0$

Implementation

Code: C code compiled with GCC 4.7.0 (64-bit). Optimized for the Sandy Bridge architechture (SSE and AVX instructions, PCLMULQDQ (carry-less multiplication instruction)).
Program code publicly available at `http://bench.cr.yp.to`.

Benchmarking: Intel Xeon E31270 3.4 GHz (Sandy Bridge) and Intel Core i5 3570 3.4 GHz (Ivy Bridge). Turbo Boost and Hyper-Threading disabled.

Implementation

Timing attacks

Protection against timing attacks is achieved through regular recoding (5-NAF).

Implementation

Timing attacks

Protection against timing attacks is achieved through regular recoding (5-NAF).

Penalties:

- Higher scalar density: $\dfrac{1}{w-1}$ against $\dfrac{1}{w+1}$ of unprotected version.
- Pre/post computation are more expensive.
- To avoid cache-timing attacks, linear passes must be executed for every point addition.

Implementation

Timing attacks

Protection against timing attacks is achieved through regular recoding (5-NAF).

Penalties:

- Higher scalar density: $\dfrac{1}{w-1}$ against $\dfrac{1}{w+1}$ of unprotected version.
- Pre/post computation are more expensive.
- To avoid cache-timing attacks, linear passes must be executed for every point addition.

Which method?

- Right-to-left halve-and-add uses multiple accumulators, hence two linear passes per addition are necessary.
- Half-trace uses look-up tables and therefore needs linear passes.

Left-to-right Double-and-add is more promising.

Implementation

Timing attacks

Protection against timing attacks is achieved through regular recoding (5-NAF).

Penalties:

- Higher scalar density: $\frac{1}{w-1}$ against $\frac{1}{w+1}$ of unprotected version.
- Pre/post computation are more expensive.
- To avoid cache-timing attacks, linear passes must be executed for every point addition.

Which method?

- Right-to-left halve-and-add uses multiple accumulators, hence two linear passes per addition are necessary.
- Half-trace uses look-up tables and therefore needs linear passes.

Left-to-right Double-and-add is more promising.

Lambda Coordinates Aftermath

One multiplication can be saved by doing doubling-and-addition and addition: $2Q + P_i + P_j$ $(17\tilde{m} + \tilde{m}_a + 8\tilde{s})$. Also, only one linear pass for two points.

Results

Scalar Multiplication

Scalar multiplication	Curve	Security	Method	SCR	Cycles
Taverne et al.	NIST-K233	112	No-GLV (τ-and-add)	no	67,800
Bos et al.	BK/FKT	128	4-GLV (double-and-add)	no	156,000
Aranha et al.	NIST-K283	128	2-GLV (τ-and-add)	no	99,200
Longa and Sica	GLS	128	4-GLV (double-and-add)	no	91,000
Taverne et al.	NIST-K233	112	No-GLV, parallel (2 cores)	no	46,500
Longa and Sica	GLS	128	4-GLV, parallel (4 cores)	no	61,000
Bernstein	Curve25519	128	Montgomery ladder	yes	194,000
Hamburg	Montgomery	128	Montgomery ladder	yes	153,000
Longa and Sica	GLS	128	4-GLV (double-and-add)	yes	137,000
Bos et al.	Kummer	128	Montgomery ladder	yes	117,000
This work	GLS	128	2-GLV (double-and-add) (LD)	no	117,500
			2-GLV (double-and-add) (λ)	no	93,500
			2-GLV (halve-and-add) (LD)	no	81,800
			2-GLV (halve-and-add) (λ)	no	**72,300**
			2-GLV, parallel (2 cores) (λ)	no	**47,900**
			2-GLV (double-and-add) (λ)	yes	**114,800**

Single core non-protected version: 17% and 27% faster than state-of-the-art implementations over prime and binary curves.

Results

Scalar Multiplication

Scalar multiplication	Curve	Security	Method	SCR	Cycles
Taverne et al.	NIST-K233	112	No-GLV (τ-and-add)	no	67,800
Bos et al.	BK/FKT	128	4-GLV (double-and-add)	no	156,000
Aranha et al.	NIST-K283	128	2-GLV (τ-and-add)	no	99,200
Longa and Sica	GLS	128	4-GLV (double-and-add)	no	91,000
Taverne et al.	NIST-K233	112	No-GLV, parallel (2 cores)	no	46,500
Longa and Sica	GLS	128	4-GLV, parallel (4 cores)	no	61,000
Bernstein	Curve25519	128	Montgomery ladder	yes	194,000
Hamburg	Montgomery	128	Montgomery ladder	yes	153,000
Longa and Sica	GLS	128	4-GLV (double-and-add)	yes	137,000
Bos et al.	Kummer	128	Montgomery ladder	yes	117,000
This work	GLS	128	2-GLV (double-and-add) (LD)	no	117,500
			2-GLV (double-and-add) (λ)	no	93,500
			2-GLV (halve-and-add) (LD)	no	81,800
			2-GLV (halve-and-add) (λ)	no	**72,300**
			2-GLV, parallel (2 cores) (λ)	no	**47,900**
			2-GLV (double-and-add) (λ)	yes	**114,800**

Two core non-protected version: 21% faster than state-of-the-art four-core implementation over prime curves.

Results (ongoing work)

Intel Haswell processor

Latency of PCLMULQDQ (carry-less multiplication instruction) dropped from 14 (Sandy Bridge) to 7. Point operations which require more field multiplications were benefited (eg. doubling, addition).

Scalar multiplication	Curve	Security	Method	SCR	Cycles
This work	GLS	128	2-GLV (double-and-add) (λ)	no	**49,455**
			2-GLV (halve-and-add) (λ)	no	**44,653**
			2-GLV, parallel (2 cores) (λ)	no	**29,450**
			2-GLV (double-and-add) (λ)	yes	**65,820**

Timings measured in a Core i7 4700MQ, 2.40GHz.

Results (ongoing work)

Intel Haswell processor

Latency of PCLMULQDQ (carry-less multiplication instruction) dropped from 14 (Sandy Bridge) to 7. Point operations which require more field multiplications were benefited (eg. doubling, addition).

Scalar multiplication	Curve	Security	Method	SCR	Cycles
This work	GLS	128	2-GLV (double-and-add) (λ)	no	**49,455**
			2-GLV (halve-and-add) (λ)	no	**44,653**
			2-GLV, parallel (2 cores) (λ)	no	**29,450**
			2-GLV (double-and-add) (λ)	yes	**65,820**

Timings measured in a Core i7 4700MQ, 2.40GHz.

The difference between double-and-add and halve-and-add was reduced from 24,400 cc (Sandy Bridge) to 4,800 cc.

Results (ongoing work)

Intel Haswell processor

Latency of PCLMULQDQ (carry-less multiplication instruction) dropped from 14 (Sandy Bridge) to 7. Point operations which require more field multiplications were benefited (eg. doubling, addition).

Scalar multiplication	Curve	Security	Method	SCR	Cycles
This work	GLS	128	2-GLV (double-and-add) (λ)	no	**49,455**
			2-GLV (halve-and-add) (λ)	no	**44,653**
			2-GLV, parallel (2 cores) (λ)	no	**29,450**
			2-GLV (double-and-add) (λ)	yes	**65,820**

Timings measured in a Core i7 4700MQ, 2.40GHz.

The difference between double-and-add and halve-and-add was reduced from 24,400 cc (Sandy Bridge) to 4,800 cc. The parallel version may soon achieve a speedup close to 2x.

Conclusion Remarks

The Lambda Coordinates system provides simple and efficient formulas for binary elliptic curve artithmetic. Combined with other techniques we could achieve a fast scalar multiplication.

More applications for the coordinates will be considered, **stay tuned!**

Thank you!