Performance comparisons of elliptic curve systems in software

Kenny Fong	University of Waterloo
Darrel Hankerson	Auburn University, USA
Julio López	University of Valle, Colombia
Alfred Menezes	University of Waterloo
Matt Tucker	University of Waterloo

October 2001

Goals

1. Do balanced comparisions.
2. Questions:
   (a) Which is faster, ECC over $\mathbb{F}_p$ or over $\mathbb{F}_{2^m}$?
   (b) How much faster is ECC over $\mathbb{F}_p$ in assembler than in C?
   (c) How much faster is ECC over NIST primes versus random primes?
   (d) How much faster is ECC over Koblitz curves versus random curves over $\mathbb{F}_{2^m}$?
   (e) Is ECC over OEFs significantly faster?
   (f) Can point halving be used effectively?
   (g) Performance when memory is constrained versus unconstrained?

Outline

1. Operations in ECDSA.
2. Platform characteristics.
3. The NIST curves.
   3.1 Random binary and Koblitz.
   3.2 Random prime.
4. Using efficient endomorphisms (GLV).
5. Point halving.
6. Optimal Extension Fields.

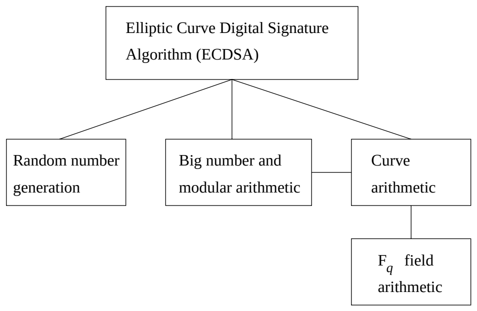
1. Elliptic curve digital signature algorithm (ECDSA)

- Signer A has domain parameters $D = (q, \text{FR}, a, b, G, n, h)$, private key d, and public key $Q = dG$. B has authentic copies of D and Q.
- To sign a message m, A does the following:
  1. Select a random integer k from $[1, n-1]$.
  2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$.
  3. Compute $e = \text{SHA-1}(m)$.
  4. Compute $s = k^{-1}\{e + dr\} \bmod n$.
  5. A's signature for the message m is (r, s).
- The computationally expensive operation is the scalar multiplication kG in step 2, for a point G which is known a priori.

ECDSA

- To verify A's signature (r, s) on m, B does:
  1. Verify that r and s are integers in $[1, n-1]$.
  2. Compute $e = \text{SHA-1}(m)$.
  3. Compute $w = s^{-1} \bmod n$.
  4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
  5. Compute $u_1G + u_2Q = (x_1, y_1)$.
  6. Compute $v = x_1 \bmod n$.
  7. Accept the signature if and only if $v = r$.
- The computationally expensive operation is the scalar multiplications u_1G and u_2Q in step 5, where only G is known a priori.

Optimizing ECC

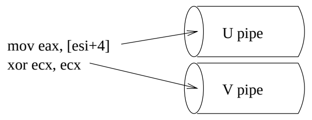

Elliptic Curve Digital Signature Algorithm (ECDSA)
Random number generation
Big number and modular arithmetic
Curve arithmetic
F_q field arithmetic

Once the field (e.g., prime or binary) and curve (e.g., random or Koblitz) are selected, speed depends largely on

1. field operations, and
2. efficient curve operations.

Big number routines (e.g., in Koblitz τ-adic NAF) are of less importance.

2. Pentium II/III characteristics



- Original Pentium used a dual pipeline.
- 8 registers (excluding flags, segment, floating point, and MMX). Integer multiplication $32 \times 32 \to 64$ bits must use a and d.
- Faster multiplication than original Pentium and MMX (latency 4 vs 9 and throughput of 1 vs 1/9).
- Better branch prediction than the original Pentium, but mispredictions are more expensive.

Compilers

- Microsoft C selected for historical reasons. Appears to give better (pipeline-friendly) sequences compared with GNU C.
- Does not honor "register" declaration, and its register allocation strategy can be weak.
- Does not process $32 \times 32 \to 64$ bit multiplication consistently.
- Assembler coding essential for odd characteristic fields.

3. NIST Recommended Elliptic Curves

- Collection of elliptic curves recommended for use with ECDSA by the US Federal Government.
- Recommended fields:

Block cipher key length	Block cipher	$\mathbb{F}_p$ $\lvert p \rvert$	$\mathbb{F}_{2^m}$ m
80	SKIPJACK	192	163
112	3-DES	224	233
128	AES Small	256	283
192	AES Medium	384	409
256	AES Large	521	571

Recommended Curves over $\mathbb{F}_{2^m}$

- Koblitz curves:

K-163	$y^2 + xy = x^3 + x^2 + 1$ over $\mathbb{F}_{2^{163}}$, cofactor 2
	$f = x^{163} + x^7 + x^6 + x^3 + 1$
K-233	$y^2 + xy = x^3 + 1$ over $\mathbb{F}_{2^{233}}$, cofactor 4
	$f = x^{233} + x^{74} + 1$
K-283	$y^2 + xy = x^3 + 1$ over $\mathbb{F}_{2^{283}}$, cofactor 4
	$f = x^{283} + x^{12} + x^7 + x^5 + 1$
K-409	$y^2 + xy = x^3 + 1$ over $\mathbb{F}_{2^{409}}$, cofactor 4
	$f = x^{409} + x^{87} + 1$
K-571	$y^2 + xy = x^3 + 1$ over $\mathbb{F}_{2^{571}}$, cofactor 4
	$f = x^{571} + x^{10} + x^5 + x^2 + 1$

- Randomly-generated curves B-{163, 233, 283, 409, 571} over each of these fields, each with cofactor 2: $y^2 + xy = x^3 + x^2 + b$.

Binary Field Arithmetic

- Timings (in μs) on a 1000 MHz Pentium III.

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$
Addition	0.032	0.039	0.041
Modular reduction	0.081	0.094	0.145
Multiplication (including reduction)			
Shift-and-add	6.11	9.66	13.25
LR comb with windows of size 4	1.06	1.92	2.40
Karatsuba	1.49	2.69	3.13
Squaring	0.19	0.24	0.31
Inversion	10.0	17.4	24.5
Inversion / Multiplication	9.5	9.1	10.2

- Addition in the binary case is especially simple (XOR).
- Squaring is by 8-to-16-bit table-lookup.
- Code is in C, except for one operation (bit scan) in inversion.

Recommended Curves over $\mathbb{F}_p$

- $y^2 = x^3 - 3x + b$, curves randomly generated and have prime order.

Curve	Prime p
P-192	$2^{192} - 2^{64} - 1$
P-224	$2^{224} - 2^{96} + 1$
P-256	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
P-384	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
P-521	$2^{521} - 1$

- The form of the prime (or reduction poly in binary case) makes reduction fast; e.g.,

Algorithm ([Solinas] fast reduction modulo $p = 2^{192} - 2^{64} - 1$)
INPUT: Integer $c = (c_5, c_4, c_3, c_2, c_1, c_0)$. OUTPUT: $c \bmod p$.

```
1. Define 192-bit integers: $s_1 = (c_2, c_1, c_0)$, $s_2 = (0, c_3, c_3)$,
   $s_3 = (c_4, c_4, 0)$, $s_4 = (c_5, c_5, c_5)$.
2. Return ($s_1 + s_2 + s_3 + s_4 \bmod p$).
```

Prime Field Arithmetic

- Compare fast reduction with Barrett:

Algorithm (Barrett reduction)
INPUT: $b > 3$, $k = \lfloor \log_b p \rfloor + 1$, $x < b^{2k}$, $\mu = \lfloor b^{2k}/p \rfloor$.
OUTPUT: $x \bmod p$.

```
1. $\hat{q} \leftarrow \lfloor \lfloor x/b^{k-1} \rfloor \cdot \mu / b^{k+1} \rfloor$.
2. $r \leftarrow (x \bmod b^{k+1}) - (\hat{q} \cdot p \bmod b^{k+1})$.
3. If $r < 0$ then $r \leftarrow r + b^{k+1}$.
4. While $r \geq p$ do: $r \leftarrow r - p$.
5. Return ($r$).
```

- b can be chosen to correspond to machine word size.
- Operations are relatively simple, and μ can be calculated once per field, but Barrett is still much slower.

Timings (in μs) for operations in the NIST prime fields

	$\mathbb{F}_{p_{192}}$[a]	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{p_{224}}$	$\mathbb{F}_{p_{256}}$
Addition	0.145	0.055	0.062	0.071
Subtraction	0.149	0.054	0.061	0.068
Modular reduction				
Barrett reduction	1.606	0.413	0.525	0.638
Fast reduction	0.191	0.097	0.122	0.256
Multiplication (including fast reduction)				
Classical	0.631[b]	0.350	0.456	0.681
Karatsuba	1.481[c]	0.825	1.100	1.413
Squaring (including fast reduction)				
Classical	—	0.300	0.394	0.600
Handbook	0.969[c]	0.425	0.544	0.794
Inversion	48.8	21.1	28.4	36.8
Inversion / Multiplication	77	60	62	54

[a]Coded primarily in C. [b]Uses a 32×32 multiply-and-add. [c]Uses a 32×32 multiply.

- Some assembler coding is essential. MSC has relatively poor register allocation strategy.
- Barrett reduction does not use the special form of the prime.

Comparison of timings for the prime and binary fields

	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{p_{224}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{p_{256}}$	$\mathbb{F}_{2^{283}}$
Addition	0.055	0.032	0.062	0.039	0.071	0.041
Modular reduction	0.097	0.081	0.122	0.094	0.256	0.145
Multiplication	0.350	1.058	0.456	1.923	0.681	2.403
Squaring	0.300	0.185	0.394	0.238	0.600	0.312
Inversion	21.1	10.0	28.4	17.4	36.8	24.5
Inversion / Mult	60.2	9.5	62.4	9.1	54.1	10.2

- Squaring is very fast in the binary case, but only 14% faster in the prime case.
- I/M differs significantly.

Elliptic Curve Arithmetic

- Coordinate choices: affine vs various projective reps. Op counts:

	Doubling	Addition (mixed)
Binary fields		
Affine	$1I, 2M$	$1I, 2M$
Projective $(X/Z, Y/Z^2)$	$4M$	$8M$
Prime fields		
Affine	$1I, 2M, 2S$	$1I, 2M, 1S$
Projective $(X/Z^2, Y/Z^3)$	$4M, 4S$	$8M, 3S$

- Many variants of simple double-and-add algorithm for kP.
  - P fixed vs P not known a priori.
  - Choices may be subject to memory constraints.
  - Signed-digit reps for k reduce the number of point additions in kP. Windowing methods.
  - Replace doubling by halving in the binary case.
  - Replace doubling by other efficiently-computable maps.

Koblitz curves

- Let $E : y^2 + xy = x^3 + ax^2 + 1$ be an elliptic curve defined over $\mathbb{F}_2$.
- (Frobenius map) Let $\tau : (x, y) \mapsto (x^2, y^2)$. $\tau^2 + 2 = (-1)^{1-a}\tau$.
- To compute kP for P in the main subgroup of $E(\mathbb{F}_{2^m})$:
  - Compute $k' = k \bmod (\tau^m - 1)/(\tau - 1)$ in $\mathbb{Z}[\tau]$.
  - Compute a τ-adic expansion of k', $\sum_{i=0}^{t} c_i \tau^i$, where $t \approx m$ and $c_i \in \{0, 1\}$.
  - $kP = k'P = \sum_{i=0}^{t} c_i \tau^i P$.
- Width-w τ-adic NAFs reduce the number of point additions.

J. Solinas, Efficient arithmetic on Koblitz curves. Designs, Codes and Cryptography, 2000.

Timings (in μs) for kP, P fixed, in ECDSA signature generation

Curve type	Memory constrained?	Fastest method	NIST curve		
			P-192	P-224	P-256
Random prime	No	Fixed-base comb (w=4, ×2)	280	406	686
	Yes	Interleave ({3, 3}-NAF)	500	780	1,250
	Yes	Binary NAF Jacobian	874	1,312	2,156
			B-163	B-233	B-283
Random binary	No	Fixed-base comb (w=5)	480	1,178	1,803
	Yes	Interleave ({3, 3}-NAF)	817	2,068	3,125
	Yes	Montgomery	1,203	3,006	4,520
			K-163	K-233	K-283
Koblitz binary	No	Fixed-base TNAF (w=6)	385	842	1,226
	Yes	TNAF	649	1,514	2,283

- In prime case, Jacobian and Chudnovsky coordinates used because of the large I/M.
- Known-point multiplications were significantly faster in the Koblitz and random prime cases.

Timings (in μs) for $kP + lQ$ in ECDSA signature verification

Curve type	Memory constrained?	Fastest method	NIST curve		
			P-192	P-224	P-256
Random prime	No	Interleave ({6, 5}-NAF)	938	1,374	2,250
	Yes	Interleave ({3, 3}-NAF)	1,064	1,562	2,562
			B-163	B-233	B-283
Random binary	No	Interleave ({6, 4}-NAF)	1,466	3,582	5,385
	Yes	Interleave ({3, 3}-NAF)	1,683	4,206	6,274
			K-163	K-233	K-283
Koblitz binary	No	Interleave ({6, 5}-TNAF)	792	1,731	2,548
	Yes	Interleave ({3, 3}-TNAF)	1,034	2,380	3,509

- Smaller differences in Koblitz binary and random prime times for $kP + lQ$ (Q not known a priori); both faster than random binary.

Timings (in μs) of the fastest methods for point multiplication kP and for $kP + lQ$ (P fixed and Q not known a priori) on P-192

Point multiplication method	Field arithmetic primarily in asm	Barrett[a] reduction	Field arithmetic primarily in C
For kP:			
Fixed-base comb ($w = 4$)	280	500	624
Interleave ({3, 3}-NAF)	500	938	1,250
For $kP + lQ$:			
Interleave ({6, 5}-NAF)	938	1,720	2,220
Interleave ({3, 3}-NAF)	1,064	1,906	2,468

[a] Fast reduction is replaced by an assembler version of Barrett.

- Barrett column can be interpreted as rough timings for ECDSA operations over a random 192-bit prime.
- Significant performance improvements from asm coding in field ops.

4. Using efficient endomorphisms

GLV observed that an endomorphism may be used to reduce the number of doubles (even if a Koblitz-like expansion is not efficient).

Example (WAP)

- Let $p \equiv 1 \pmod 3$, $E : y^2 = x^3 + b$, and let $\beta \in \mathbb{F}_p$ be an element of order 3.
- $\phi : (x, y) \mapsto (\beta x, y)$ is an endomorphism.
- Computing ϕ requires only 1 field multiplication.
- $|\phi| = 1$.

1. Gallant, Lambert, and Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms, 1999.
2. Park, Jeong, and Kim. An alternate decomposition of an integer for faster point multiplication on certain elliptic curves, 2001.

Using efficient endomorphisms (2/2)

Basic idea:

- Let $G \in E(\mathbb{F}_p)$ be a point of prime order n.
- ϕ acts on $\langle G \rangle$ by multiplication: $\phi P = \lambda P$, where λ is a root (modulo n) of the characteristic polynomial of ϕ. ($\lambda^2 + \lambda \equiv -1 \pmod{n}$ in the example.)
- To compute kP:
  - Write $k \equiv k_1 + k_2\lambda \pmod{n}$ where $k_i \in [0, \sqrt{n}]$. (This can be done efficiently.)
  - $kP = k_1 P + k_2 \lambda P = k_1 P + k_2 \phi(P)$, which can be computed via interleaving.
- Approx half the doubles are eliminated. Cost of finding k_i negligible.

Timings for the WAP curve

- $p = 2^{160} - 229233$; curve $y^2 = x^3 + b$ over $\mathbb{F}_p$ with approx 2^{160} points.

Method	Time
For lQ, Q unknown:	
Binary	648
Interleave using ϕ	480
Interleave using ϕ & 4-NAF	385
For $kP + lQ$:	
Interleave ($\{6, 4\}$-NAF)	625
Interleave using comb, ϕ & 4-NAF	505

- Fast reduction.
- Less useful on kP if precomputation can be used.
- Comb method on kP can be effectively combined with the GLV method on lQ.
- Finding $k \equiv k_1 + k_2\lambda$ is 0.7% of the time for performing kP.

5. Point halving for curves over binary fields

- Doubling in affine: seek $2P = (x_2, y_2)$ from $P = (x, y)$.

Let $\lambda = x + y/x$. Calculate:

$$\begin{array}{lcl} x_2 = x^2 + b/x^2 & & x_2 = \lambda^2 + \lambda + a \\ y_2 = x^2 + \lambda x_2 + x_2 & \text{or} & y_2 = x^2 + \lambda x_2 + x_2 \\ \text{(2 mul, 1 mul by } b\text{, 1 inv)} & & \text{(2 mul, 1 inv)} \end{array}$$

- Halving: seek $P = (x, y)$ from $2P = (x_2, y_2)$. Basic idea: solve

$$\begin{array}{ll} x_2 = \lambda^2 + \lambda + a & \text{for } \lambda \\ y_2 = x^2 + \lambda x_2 + x_2 & \text{for } x \end{array}$$

1. E. Knudsen, Elliptic scalar multiplication using point halving, Asiacrypt ’99.
2. R. Schroeppel, Elliptic curve point ambiguity resolution apparatus and method, patent application, 2000.

Facts

1. $\mathrm{Trace}(c) = c + c^2 + \cdots + c^{2^{m-1}} \in \{0, 1\}$.
2. The NIST random binary curves all have $\mathrm{Trace}(a) = 1$. $\mathrm{Trace}(x(kG)) = \mathrm{Trace}(a)$ for generator G.

Halving for the trace 1 case

1. Solve
$$\widehat{\lambda}^2 + \widehat{\lambda} = x_2 + a$$
obtaining $\widehat{\lambda} = \lambda$ or $\widehat{\lambda} = \lambda + 1$.
2. Since $y_2 = x^2 + \lambda x_2 + x_2$, consider
$$\widehat{x}^2 = (\widehat{\lambda} + 1)x_2 + y_2$$
$\mathrm{Trace}(x^2) = \mathrm{Trace}(x) = \mathrm{Trace}(a) = 1$, so $\mathrm{Trace}((\widehat{\lambda} + 1)x_2 + y_2)$ identifies λ.
3. Find $x = \sqrt{x_2(\lambda + 1) + y_2}$.

Halving: $(x_2, y_2) \to (x, \lambda = x + y/x)$ where $2(x, y) = (x_2, y_2)$; y may be recovered via

$$\lambda x = x^2 + y \implies y = \lambda x + x^2 \quad (\approx 1 \text{ field mult})$$

Algorithm (point halving) Input: (x_2, λ_2) or (x_2, y_2). Output: $(x, \lambda = x + y/x)$ where $2(x, y) = (x_2, y_2)$.

Steps	Cost
1. Solve $\widehat{\lambda}^2 + \widehat{\lambda} = x_2 + a$ for $\widehat{\lambda}$.	$\approx 3/4$ field mult
2. Find $T = x_2(\widehat{\lambda} + \lambda_2 + x_2 + 1)$	≈ 1 field mult
or $T = x_2(\widehat{\lambda} + 1) + y_2$	
3. If Trace$(T) = 1$ then $\lambda = \widehat{\lambda}$, $x = \sqrt{T}$	Trace $\approx$ free
else $\lambda = \widehat{\lambda} + x_2$, $x = \sqrt{T + x_2}$.	root $\approx 1/2$ field mult
4. Return(x, λ).	

Conversion to affine $(x, \lambda) \to (x, y)$ is ≈ 1 field mult.
(Doubling in projective ≈ 4 field mults.)

Calculating kP by double-and-add and halve-and-add

Algorithm (double-and-add, right to left) Input: point P and scalar k. Output: kP.

```
1. $Q \leftarrow 0$.
2. For $i$ from 0 to $\log k$ do
   2.1 If $k_i = 1$ then $Q \leftarrow Q + P$.
   2.2 $P \leftarrow 2P$.
3. Return$(Q)$.
```

- Double-and-add is easily converted to left-to-right.
- Window NAF methods reduce the number of additions.

Algorithm (halve-and-add, right to left) Input: point P and scalar k. Output: kP.

```
1. (Precomputation) Solve quadratic equations (44 field elements for
   B-163). Build table of 16 or 64 multiples of $\sqrt{x}$.
2. (Transform $k$) Solve
       $k = k_t 2^t + \cdots + k_0 = k'_t/2^t + \cdots + k'_1/2 + k'_0 \pmod n$
   for $k'$; i.e.,
       $2^t k \bmod n = k'_0 2^t + \cdots + k'_t$
3. $Q \leftarrow 0$.
4. For $i$ from 0 to $t$ do
   4.1 If $k'_i = 1$ then $Q \leftarrow Q + P$.
   4.2 $P \leftarrow P/2$.
5. Return$(Q)$.
```

Timings for B-163

Field operations	
multiplication	1.06
inversion	10.05
I/M	9.5
sqrt (16-point table)	0.63
sqrt64 (64-point table)	0.46
solve QE	0.89

Curve operations	
halve with sqrt	2.24
halve with sqrt64	2.16
Scalar multiplication kP	
Montgomery	1203
halving with NAF and sqrt	1057
halving with NAF and sqrt64	1011
4-NAF	1178
Comb (known-point, w=5)	480

1. Conversion (x, λ) to affine (costing ≈ 1 field mult) is done only when a point addition is required.
2. Halving is 12–16% faster than Montgomery, but requires storage for at least 60 field elements.
3. Halve-and-add can use NAF to reduce the number of additions.
4. Halving operates on (x, λ) or (x, y), not on projective reps. Algorithm is right-to-left.
5. Halving could be used efficiently in left-to-right algs if I/M small.

6. Optimal Extension Fields (OEF)

Bailey and Paar, CRYPTO ’98 and J. Cryptology 2001.

- $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f)$ for $p = 2^n \pm c$, $\log c \le n/2$, and $f = x^m - \omega$ irreducible.
- Type 1: $c = 1$; e.g., $m = 6$, $p = 2^{31} - 1$, $f = x^6 - 7$.
  Type 2: $\omega = 2$; e.g., $m = 5$, $p = 2^{32} - 5$, $f = x^5 - 2$.
- p can be chosen to fit in a single machine word, and $\mathbb{F}_{p^m}$ arithmetic can be performed via operations in the $\mathbb{F}_p$. Field inversion is fast compared with that in $\mathbb{F}_q$ if prime $q \approx p^m$.
- Subfield multiplication: for $p = 2^n - c$, to find $ab \bmod p$,

$$ab = d_1 2^n + d_0 \equiv d_1 c + d_0$$
$$d_1 c = e_1 2^n + e_0 \equiv \underbrace{e_1 c}_{f_0} + e_0$$

  so $ab = d_0 + e_0 + f_0 \bmod p$. (Type 1 preferable here.)

Multiplication in OEFs

$$C(x) = A(x)B(x) = \sum_{i+j<m} A_i B_j x^{i+j} + \sum_{i+j>m} A_i B_j x^{i+j-m}$$

- $x^m \equiv \omega$, so $C(x) = \sum C_k x^k$ where

$$C_k = \sum_{i=0}^{k} A_i B_{k-i} + \omega \sum_{i=k+1}^{m-1} A_i B_{m+k-i}$$

- Expensive to perform the reductions in the subfield ops.
- Bailey and Paar present a Karatsuba-style method to trade mults for additions. For $m = 6$, 36 subfield mults and 25 adds are replaced by 18 mults and 59 additions.
- Fastest: calculate C_k with a multiply-and-accumulate strategy using 2 or more registers, minimizing reductions. Multiplication similar to that for $\mathbb{F}_q$ for $q \approx p^m$ with fast reduction.
- Type 2 ($\omega = 2$) preferable here.

Comparisons among field operations

	$\mathbb{F}_{p192}$	OEF: $m = 6$ $p = 2^{31} - 1$	$\mathbb{F}_{2^{163}}$	OEF: $m = 5$ $p = 2^{32} - 5$
add	.055	.050	.032	.048
mul	.350	.334	1.058	.250
inv	21.062	2.672	10.048	1.672
I/M	60	8.0	9.5	6.7

$\mathbb{F}_{2^{163}}$ almost entirely in C; others use assembler.

- I/M determines if projective coordinates are preferred for curve arithmetic. For curves over odd characteristic fields:

	affine	projective	
double	$I{+}2M{+}2S$	$4M{+}4S$	
add	$I{+}2M{+}S$	$8M{+}3S$	(mixed)

Inversion in OEFs (Itoh & Tsujii)

Given $A \in \mathbb{F}_{p^m}$ and $r = \frac{p^m-1}{p-1} = p^{m-1} + \cdots + p + 1$, find

$$A^{-1} = (A^r)^{-1} A^{r-1}.$$

Steps

1. Compute $A^{r-1} = A^{p^{m-1}+\cdots+p}$.
2. $A^r = A^{r-1}A \in \mathbb{F}_p$.
3. Find $c = (A^r)^{-1}$ in $\mathbb{F}_p$.
4. $A^{-1} = cA^{r-1}$.

Remarks

- Steps 1 and 3 appear to be the expensive calculations.
- Step 2 is not a full field multiplication.
- Step 3 is inversion in $\mathbb{F}_p$, which is relatively fast.
- Step 1 can be done with a few field multiplications.

Cost of inversion in OEFs

1. Have $m \mid p-1$ where $f(x) = x^m - \omega$. Then

$$A^{p^j} = a_0 + a_1 x^{p^j} + \cdots + a_{m-1} x^{(m-1)p^j}$$

and

$$(x^i)^{p^j} \equiv \omega^q x^i \pmod{f(x)}$$

2. Use of an addition chain finds A^{r-1} in $\lfloor \log(m-1) \rfloor + H(m-1) - 1$ multiplications and $\lfloor \log(m-1) \rfloor + H(m-1)$ applications of the Frobenius map.

3. Example: $m = 5, r-1 = p^4+p^3+p^2+p = [(p+1)(p^2) + (p+1)]p$.

$$A^{r-1} = ((A \cdot A^p)^{p^2} \cdot (AA^p))^p \quad \text{(2 mul, 3 Frobenius)}$$

Cost of inversion in OEFs (2/2)

4. Use binary EEA to find $(A^r)^{-1}$ in $\mathbb{F}_p$, requiring $\approx$ 40% of the total time for inversion.

5. Lim and Hwang favor an EEA-like inversion alg. Their subfield inversion appears very fast; however, their field inversion times are slower.

---

1. Daniel V. Bailey and Christof Paar. Optimal extension fields for fast arithmetic in public-key algorithms, CRYPTO ’98.
2. Daniel V. Bailey and Christof Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography, J. Cryptology, 2001.
3. Chae Hoon Lim and Hyo Sun Hwang. Fast implementation of elliptic curve arithmetic in $GF(p^n)$.

Comparisons among point operations

		OEF: $m = 6$		OEF: $m = 5$
	P-192	$p = 2^{31} - 1$	K-163	$p = 2^{32} - 5$
double	3.2 (22.8)[a]	3.0 (4.3)[a]	5.4 (12.8)[a]	2.3 (3.0)[a]
add	4.1 (22.6)[a]	4.3 (4.0)[a]	10.2 (13.0)[a]	3.1 (2.8)[a]
kP	280	235	385	156
$kP + lQ$	938	813	792	546

[a] Affine coordinates.

- Affine addition faster than projective.
- Müller (Efficient point multiplication for elliptic curves over special optimal extenstion fields) combines the ideas of GLV and OEF.
- Kobayashi et. al (Fast elliptic curve algorithm combining frobenius map and table reference to adapt to higher characteristic, Eurocrypt ’99) present Koblitz-like speedups.

7. Questions

- What’s I/M on this platform? Can halving be applied more widely?
- Code for binary fields written almost entirely in C. How much can be obtained by coding in assembler?
- The Pentium II/III has wide registers (the “multimedia” and floating-point) which can be exploited.