# *Implementing Elliptic Curve Cryptography (a narrow survey)*

***Institute of Computing – UNICAMP***
***Campinas, Brazil***
***April 2005***

Darrel Hankerson
Auburn University

**Objective:** sample selected topics of practical interest.

**Talk will favor:**

- Software solutions on general-purpose processors rather than dedicated hardware.
- Techniques with broad applicability.
- Methods targeted to standardized curves.

**Goals:**

- Present proposed methods in context.
- Limit coverage of technical details (but “implementing” necessarily involves platform considerations).

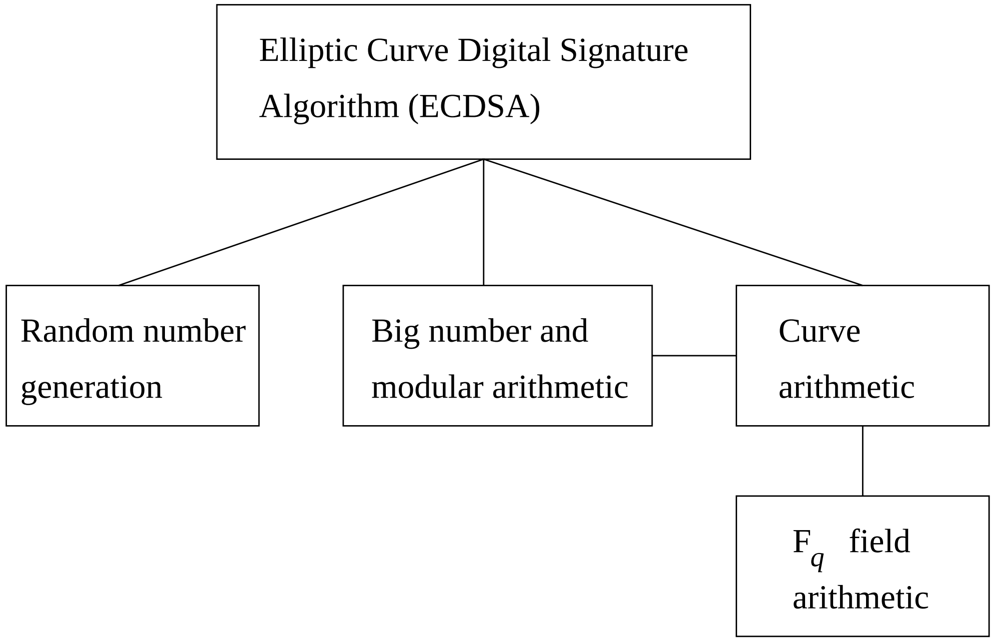
# Focus: higher-performance processors

“Higher-performance” includes processors commonly associated with workstations, but also found in surprisingly small portable devices.

Sun and IBM workstations
SPARC or Intel x86 (Pentium)
256 MB – 8 GB
0.5 GHz – 3 GHz
heats entire building


RIM pager circa 1999
Intel x86 (custom 386)
2 MB “disk”, 304 KB RAM
10 MHz, single AA battery
fits in shirt pocket

# Optimizing ECC




General categories of optimization efforts:

1. Field-level optimizations.
2. Curve-level optimizations.
3. Protocol-level optimizations.

Constraints: security requirements, hardware limitations, bandwidth, interoperability, and patents.

1. Field-level optimizations.
   - Choose fields with fast multiplication and inversion.
   - Use special-purpose hardware (cryptographic coprocessors, DSP, floating-point, SIMD).
2. Curve-level optimizations.
   - Reduce the number of point additions (windowing).
   - Reduce the number of field inversions (projective coords).
   - Replace point doubles (endomorphism methods).
3. Protocol-level optimizations.
   - Develop efficient protocols.
   - Choose methods and protocols that favor your computations or hardware.

# Context: Protocols such as ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the DSA.

Provides data origin authentication, data integrity, and non-repudiation.

| | |
|---|---|
| Jan 1999 | ECDSA formally approved as an ANSI standard – ANSI X9.62. |
| Jan 2000 | ECDSA formally approved as a US Federal Gov standard – FIPS 186-2. |
| Aug 2000 | ECDSA formally approved as an IEEE standard – IEEE 1363-2000 |
| Feb 2005 | NSA recommends algs for securing US Gov sensitive but unclassified data: ECDSA selected for authentication. |

# ECDSA Signature Generation

Signer $A$ has domain parameters $D$ (consisting of the curve, field, base point $G$, etc.), 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 $e =$ 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

$$kG = \underbrace{G + G + \cdots + G}_{k \text{ times}}$$

  in step 2, for a point $G$ which is known a priori.

# ECDSA Verification

- 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 $u_1 G + u_2 Q = (x_1, y_1)$.
  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 $u_1 G$ and $u_2 Q$ in step 5, where only $G$ is known a priori.

**US National Security Agency (NSA) and ECC**

**Fall 2003**

NSA obtains licensing rights to Menezes-Qu-Vanstone (MQV). Covers curves over $\mathbb{F}_p$ for 256-bit or larger $p$.

**February 2005, at the RSA conference**

NSA presents strategy and recommendations for securing US Gov sensitive and unclassified communications.

“Suite B” protocols for key agreement and authentication are ECC only: ECMQV and ECDH for key agreement, and ECDSA for auth.

## Example: BlackBerry 2-way pager

- Marketing demands 256-bit AES.
- Key size comparison.

| Symmetric cipher key length | Example algorithm | ECC key length | RSA/DL key length |
|---|---|---|---|
| 80 | SKIPJACK | 160 | 1024 |
| 112 | Triple-DES | 224 | 2048 |
| 128 | AES-Small | 256 | 3072 |
| 192 | AES-Medium | 384 | 8192 |
| 256 | AES-Large | 512 | 15360 |

- ECC especially attractive here.

# Deployment Notes...

- Timings on BlackBerry 7230 for 128-bit security.

| | ECC (256) | RSA (3072) | DH (3072) |
|---|---|---|---|
| Key generation | 166 ms | Too long | 38 s |
| Encrypt or verify | 150 ms | 52 ms | 74 s |
| Decript or sign | 168 ms | 8 s | 74 s |

Source: Herb Little, RIM.

- BlackBerry EC algorithms.

| Facility | Protocol |
|---|---|
| Over the Air (OTA) Provisioning | ECSPEKE[a] (ECDH-512) |
| Policy Authentication | ECDSA |
| OTA Re-key | ECMQV |

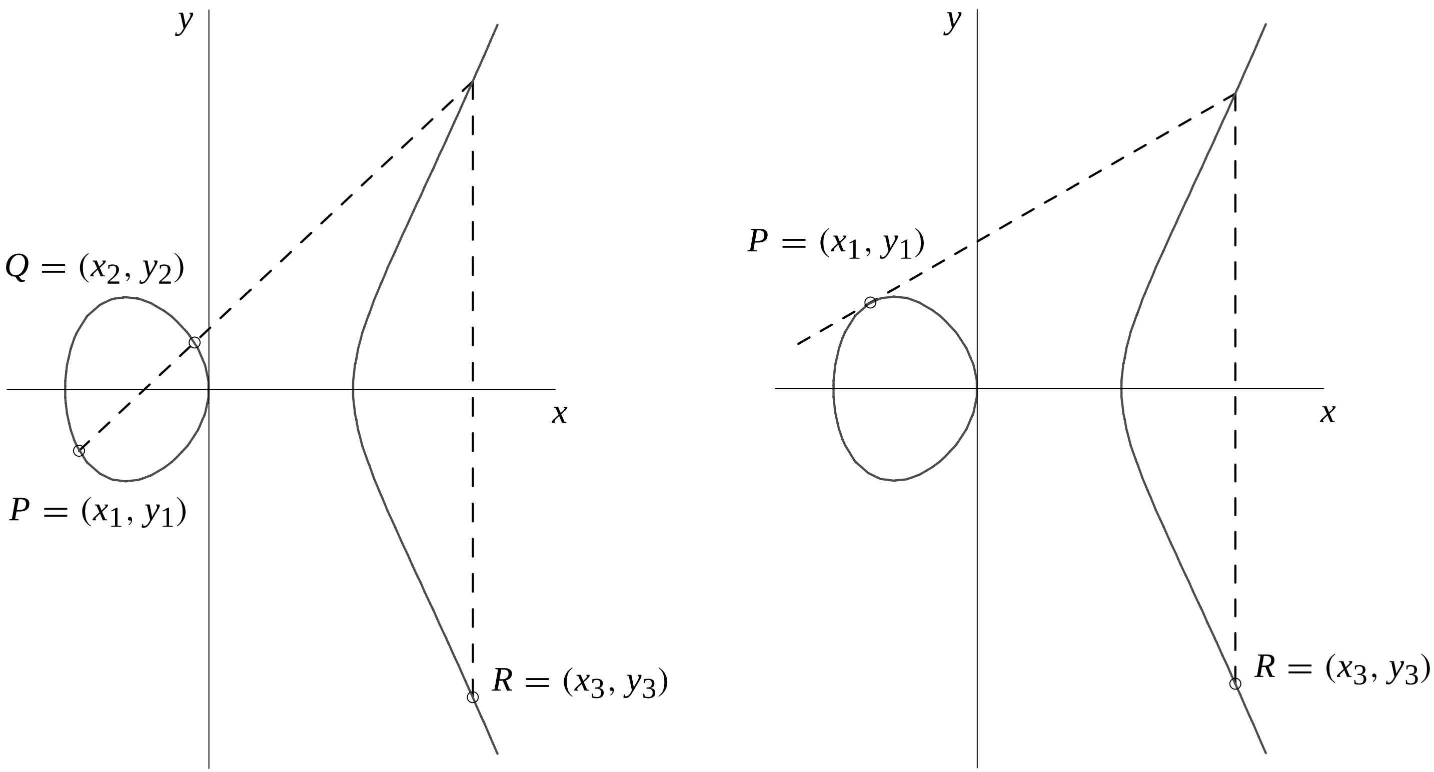
[a]Simple Password Exponential Key Exhange.

- RSA-1024 used for code signing (for verify speed).

# Topic I

# *Prerequisites*

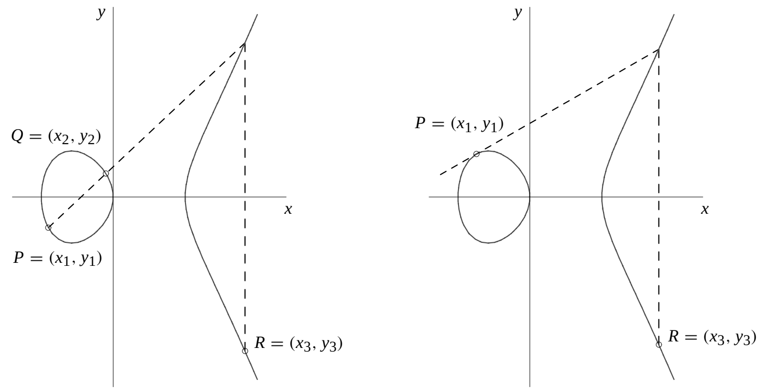
1. Interested in elliptic curves in simplified Weierstrass forms
$$\begin{aligned} y^2 &= x^3 + ax + b, \quad \text{char } K \neq 2,3 \\ y^2 + xy &= x^3 + ax^2 + b, \quad \text{char } K = 2. \end{aligned}$$
over finite fields $K$.
2. $E(K) = \{(x,y) \in K \times K \mid (x,y) \text{ solves the equation}\} \cup \{\infty\}$.
3. The chord-and-tangent rule make $E(K)$ into an abelian group with the *point at infinity* $\infty$ as the identity.
4. Scalar (or point) multiplication is the operation
$$kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$$
where $k$ is an integer.




(a) Addition: $P + Q = R$.

(b) Doubling: $P + P = R$.

Geometric addition and doubling of elliptic curve points




## Addition and Doubling

Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$.

| op | $\lambda$ | $x$ | $y$ |
|---|---|---|---|
| char $K \notin \{2,3\}$, curve: $y^2 = x^3 + ax + b$ | | | |
| $-P$ | | $x_1$ | $-y_1$ |
| $P+Q$ | $(y_2 - y_1)/(x_2 - x_1)$ | $\lambda^2 - x_1 - x_2$ | $\lambda(x_1 - x) - y_1$ |
| $2P$ | $(3x_1^2 + a)/2y_1$ | $\lambda^2 - 2x_1$ | $\lambda(x_1 - x) - y_1$ |
| char $K = 2$, curve: $y^2 + xy = x^3 + ax^2 + b$ | | | |
| $-P$ | | $x_1$ | $x_1 + y_1$ |
| $P+Q$ | $(y_1 + y_2)/(x_1 + x_2)$ | $\lambda^2 + \lambda + x_1 + x_2 + a$ | $\lambda(x_1 + x) + x + y_1$ |
| $2P$ | $x_1 + y_1/x_1$ | $\lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2}$ | $x_1^2 + \lambda x + x$ |

Simple double-and-add approach:

1. Write $k = \sum_{i=0}^{t-1} k_i 2^i$ for $k_i \in \{0,1\}$.
2. $Q \leftarrow \infty$.
3. For $i$ from $t-1$ to 0 do
   - 3.1 $Q \leftarrow 2Q$.
   - 3.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
4. Return $Q$.

Analysis: $t$ point doubles and expected $t/2$ additions.

Improving the performance:

- Reduce the number of point additions (windowing).
- Reduce the number of field inversions (projective coordinates).
- Replace point doubles (efficient endomorphisms).
- Use curves over fields where fast arithmetic is available.

Simple double-and-add approach has $t$ point doublings and an expected $t/2$ additions.

Reduce adds by writing $k$ in *non-adjacent form* (NAF):

$$\text{NAF}(k) = \sum_{i=0}^{t} k_i 2^i, \text{ where } k_i \in \{0, \pm 1\} \text{ and } k_{i+i} k_i = 0.$$

1. Find $\text{NAF}(k) = \sum_{i=0}^{t} k_i 2^i$. Set $Q \leftarrow \infty$.
2. For $i$ from $t$ to 0 do
   - 2.1 $Q \leftarrow 2Q$.
   - 2.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
   - 2.3 If $k_i = -1$ then $Q \leftarrow Q - P$.
3. Return $Q$.

Analysis: $t$ point doublings and an expected $t/3$ additions.

NAF reduces the number of required point additions. Can be generalized to a *width-$w$ NAF*.

1. $k = \sum_{i=0}^{t} k_i 2^i$, where $k_i \in \{0, \pm 1, \pm 3, \ldots, \pm 2^{w-1} - 1\}$.
2. Among $w$ consecutive digits, at most one is nonzero.
3. $t \leq \lfloor \log_2 k \rfloor + 1$.
4. Density is $1/(w+1)$. A NAF is a width-2 NAF.

Example: 13 (base 10) has expansions

$$\underbrace{(1,0,-1,0,1)}_{\text{NAF}} = \underbrace{(1,1,0,1)}_{\text{binary}} = 13_{10} = 1 \cdot 2^4 + (-3) \cdot 2^0 = \underbrace{(1,0,0,0,-3)}_{\text{width-3 NAF}}.$$

Analysis: $kP$ by width-$w$ NAF has $t$ doublings and an expected $t/(w+1)$ additions.

Calculting the $w$-NAF is inexpensive.

- Solinas [DCC 2000] gives alg using only simple bit operations.
- Digits of $w$-NAF $k = \sum k_i 2^i$ produced right-to-left ($k_0$ first).
- Some point mult algs want to process left-to-right, so typically digits of $w$-NAF are stored.

Avanzi [SAC 2004] gives (optimal) left-to-right calculation of $w$-NAF variant. See also Muir and Stinson [CACR Tech Rep].

Montgomery's method is a useful benchmark for point multiplication algorithms for curves

$$y^2 + xy = x^3 + ax^2 + b$$

over binary fields. (Due to López and Dahab and based on an idea of Montgomery.)

- Let $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ with $Q_1 \neq \pm Q_2$.
- Let $Q_1 + Q_2 = (x_3, y_3)$ and $Q_1 - Q_2 = (x_4, y_4)$.
- Then

$$x_3 = x_4 + \frac{x_2}{x_1 + x_2} + \left(\frac{x_2}{x_1 + x_2}\right)^2.$$

- Thus, the $x$-coordinate of $Q_1 + Q_2$ can be computed from the $x$-coordinates of $Q_1$, $Q_2$ and $Q_1 - Q_2$.

$$(\underbrace{k_{t-1}k_{t-2}\cdots k_{t-j}}\,\underbrace{k_{t-(j+1)}}\,k_{t-(j+2)}\cdots k_1k_0)_2P$$

$$\downarrow \qquad\qquad \downarrow$$

$$[lP,(l+1)P] \rightarrow \boxed{\begin{array}{ll} [2lP, lP+(l+1)P], & \text{if } k_{t-(j+1)} = 0 \\ [lP+(l+1)P, 2(l+1)P], & \text{if } k_{t-(j+1)} = 1 \end{array}}$$

- Each iteration requires one doubling and one addition.
- Possesses natural resistance to some side-channel attacks.
- No extra storage.
- Produces only the $x$-coordinate of the result, sufficient for ECDSA.

## *Obtaining the $y$-coordinate*

Two methods to find $y$-coordinate from a Montgomery multiplication:

1. (Direct) After the last iteration, have the $x$-coordinates of $kP = (x_1, y_1)$ and $(k+1)P = (x_2, y_2)$. $y$-coordinate of $kP$ can be recovered as:

$$y_1 = x^{-1}(x_1 + x)\left[(x_1 + x)(x_2 + x) + x^2 + y\right] + y.$$

   (Derived using the addition formula for $x$-coord $x_2$ of $(k+1)P$ from $kP = (x_1, y_1)$ and $P = (x, y)$.)

2. (Point compression method) Guess at the $y$-coordinate of $kP$ (e.g., solve quadratic $y^2 + x_1 y = x_1^2 + a x_1^2 + b$), obtaining $\tilde{y}_1$.
   Compute the $x$-coord of $(x_1, \tilde{y}_1) + (x, y)$. If this agrees with the $x$-coord of $(k+1)P$ then set $y_1 = \tilde{y}_1$; otherwise set $y_1 = \tilde{y}_1 + x_1$.

Let $c, d$ be positive integers. Define equivalence relation $\sim$ on $K^3 \backslash \{(0,0,0)\}$ by

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \quad \text{if} \quad (X_1, Y_1, Z_1) = (\lambda^c X_2, \lambda^d Y_2, \lambda Z_2)$$

for some $\lambda \in K^*$. An equivalence class is called a *projective point*.

| Projective form | Relation | Name |
|---|---|---|

Curve: $y^2 = x^3 + ax + b$

| | | |
|---|---|---|
| $Y^2Z = X^3 + aXZ^2 + bZ^3$ | $(X/Z, Y/Z)$ | projective |
| $Y^2 = X^3 + aXZ^4 + bZ^6$ | $(X/Z^2, Y/Z^3)$ | Jacobian |

Curve: $y^2 + xy = x^3 + ax^2 + b$

| | | |
|---|---|---|
| $Y^2Z + XYZ = X^3 + aX^2Z + bZ^3$ | $(X/Z, Y/Z)$ | projective |
| $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$ | $(X/Z^2, Y/Z^3)$ | Jacobian |
| $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ | $(X/Z, Y/Z^2)$ | López-Dahab (LD) |

Field inversion is typically expensive relative to field multiplication.

- Projective coordinates reduce the number of field inversions in point arithmetic.
- Point addition in affine for $\mathbb{F}_{2^m}$:

$$\lambda \leftarrow (y_1+y_2)/(x_1+x_2)$$
$$x \leftarrow \lambda^2 + \lambda + x_1 + x_2 + a, \quad y \leftarrow \lambda(x_1 + x) + x + y_1$$

- Projective $(X:Y:Z) = (X_1:Y_1:Z_1) + (X_2:Y_2:1)$:

$$A \leftarrow Y_2 Z_1^2 + Y_1,\ B \leftarrow X_2 Z_1 + X_1,\ C \leftarrow Z_1 B,\ D \leftarrow B^2(C + aZ_1^2),$$
$$Z \leftarrow C^2,\ E \leftarrow AC,\ X \leftarrow A^2 + D + E,\ F \leftarrow X + X_2 Z,$$
$$G \leftarrow (X_2 + Y_2)Z^2,\ Y \leftarrow (E + Z)F + G.$$

Affine vs projective field operation counts:

| | Doubling | Addition (mixed) |
|---|---|---|
| *Curve* $y^2 + xy = x^3 + ax^2 + b$, $a \in \{0,1\}$ | | |
| Affine | $1I$, $2M$ | $1I$, $2M$ |
| López-Dahab $(X/Z, Y/Z^2)$ | $4M$ | $8M$ |
| *Curve* $y^2 = x^3 + ax + b$, $a = -3$ | | |
| Affine | $1I$, $2M$, $2S$ | $1I$, $2M$, $1S$ |
| Jacobian $(X/Z^2, Y/Z^3)$ | $4M$, $4S$ | $8M$, $3S$ |

$I =$ inversion, $M =$ multiplication, $S =$ squaring

Rough estimate of threshold $I/M$ in binary case: $kP$ by non-adjacent form

$$\underbrace{I + 2M + \frac{1}{3}(I + 2M) = D + \frac{1}{3}A}_{\text{affine}} \le \underbrace{D_{\text{proj}} + \frac{1}{3}A_{\text{proj}} = 4M + \frac{1}{3}8M}_{\text{projective}}$$

gives $I \le 3M$.

**Example** $kP$ for the NIST random binary curve B-163 over field $\mathbb{F}_{2^{163}} = \mathbb{F}_2[z]/(z^{163}+z^7+z^6+z^3+1)$.

| Method | Coordinates | $w$ | Points stored | EC operations $A$ | EC operations $D$ | Field operations[a] $M$ | $I$ | $I/M{=}5$ | $I/M{=}8$ |
|---|---|---|---|---|---|---|---|---|---|
| *Unknown point ($kP$, on-line precomputation)* | | | | | | | | | |
| Binary | affine | 0 | 0 | 81 | 162 | 486 | 243 | 1701 | 2430 |
| | Projective | 0 | 0 | 81 | 162 | 1298 | 1 | 1303 | 1306 |
| Binary NAF | affine | 0 | 0 | 54 | 162 | 432 | 216 | 1512 | 2160 |
| | projective | 0 | 0 | 54 | 162 | 1082 | 1 | 1087 | 1090 |
| Window NAF | affine | 4 | 3 | 35 | 163 | 396 | 198 | 1386 | 1980 |
| | projective | 4 | 3 | 3[b]+32 | 163 | 914 | 5 | 939 | 954 |
| Montgomery | affine | – | 0 | 162[c] | 162[d] | 328 | 325 | 1953 | 2928 |
| | projective | – | 0 | 162[c] | 162[d] | 982 | 1 | 987 | 990 |

[a]Right columns give costs in terms of field mults for $I/M = 5$ and $I/M = 8$.
[b]Affine. [c]Addition for Montgomery. [d]$x$-coordinate only.

Required point doubles limit the improvement via $w$-NAFs.

# Appendix: NIST curves over binary fields

1. Koblitz curves:

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

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

Curves $y^2 = x^3 - 3x + b$ 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 | $2^{521} - 1$ |

Special form of the prime speeds reduction.

# Appendix: Basic facts for curves over prime fields

1. There are about $2p$ different elliptic curves over $\mathbb{F}_p$.
2. $E(\mathbb{F}_p)$ is an abelian group with identity $\infty$.
3. (*Hasse's Theorem*) The number of points on the elliptic curve is $\#E(\mathbb{F}_p) = p+1-t$ where $|t| \leq 2\sqrt{p}$; that is,
$$p+1-2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p+1+2\sqrt{p}.$$
Hence, $\#E(\mathbb{F}_p) \approx p$.
4. $\#E(\mathbb{F}_p)$ can be computed in polynomial time using *Schoof's algorithm*.
5. $E(\mathbb{F}_p)$ is isomorphic to $\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$ where $n_2$ divides both $n_1$ and $p-1$.

Corresponding facts hold if $\mathbb{F}_p$ is replaced by any finite field.

# Topic II

# ***Endomorphism methods***

# Reducing the cost of doubling

1. Use projective coordinates when $I/M$ is large. Eliminates most field inversions.

2. Use precomputation if the point is known in advance (e.g., in signature generation) and the additional memory requirement is acceptable.

3. Replace doubling by halving in the binary case.

4. Replace (some) doublings by other efficiently computable maps.

## Replace doublings: efficient endomorphisms

1. (multiplication by $m$ map) $[m] : P \mapsto mP$. Special case: $P \mapsto -P$.
2. ($q$th power map) $\phi : (x, y) \mapsto (x^q, y^q)$ for $E$ defined over $\mathbb{F}_q$. Particular case: Koblitz curves, $q = 2$.
3. Let $p \equiv 1 \pmod 3$. Consider $E : y^2 = x^3 + b$ defined over $\mathbb{F}_p$. If $\beta \in \mathbb{F}_p$ is of order $3$, then

$$\phi : (x, y) \mapsto (\beta x, y)$$

   is an endomorphism.

For $q = 2$, the map in 2 is inexpensive compared with field mult, and the map in 3 is a single field mult.

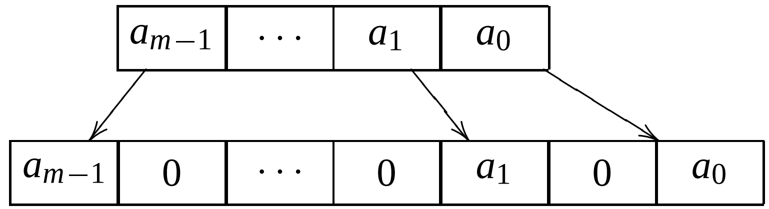
Koblitz curves: inexpensive applications of $\phi$ replace point doubles. (Point halving has similar goal.)

Example 3 allows the number of doublings to be reduced.

# Koblitz curves

Define curves over $\mathbb{F}_2$:

$$E_0: \quad y^2 + xy = x^3 + 1$$
$$E_1: \quad y^2 + xy = x^3 + x^2 + 1$$




Squaring a binary polynomial $a_{m-1}z^{m-1} + \ldots + a_1 z + a_0$.

- (Frobenius map) $\tau : (x, y) \mapsto (x^2, y^2)$.
- $\tau^2 P + 2P = \mu\tau(P)$ for $\mu = (-1)^{1-a}$ and curve points $P$.
- $\tau^2 + 2 = \mu\tau \implies \tau = \frac{\mu+\sqrt{-7}}{2}$.
- Makes sense to multiply points in $E_a(\mathbb{F}_{2^m})$ by elements of $\mathbb{Z}[\tau]$:

$$(u_l\tau^l + \cdots + u_1\tau + u_0)P = u_l\tau^l(P) + \cdots + u_1\tau(P) + u_0 P$$

Applying $\tau$ (field squaring) is inexpensive in comparison to field multiplication.

Basic idea: since field squaring is cheap, expand $k$ as $\sum k_i \tau^i$ with $|k_i|$ small and sparse (to reduce the number of required point additions).

- $\mathbb{Z}[\tau]$ is Euclidean with respect to $N : \alpha \mapsto \alpha\overline{\alpha}$.
- Finding a width-$w$ $\tau$-NAF is analogous to ordinary width-$w$ NAF: for odd $r_0 + r_1\tau$, the element of the equivalence class (mod $\tau^w$) is subtracted, and the result is divisible by $\tau^w$.

  Example: representatives are $\alpha_u = u \bmod \tau^w$. If $w = 3$ and $a = 0$, then $\alpha_1 = 1$, $\alpha_3 = \tau + 1$, and
  $$5 = -\tau^5 + 0\tau^4 + 0\tau^3 + 0\tau^2 + 0\tau^1 - (\tau + 1)\tau^0$$
  gives a width-3 $\tau$-NAF of $5$. Then $5P = -\tau^5\alpha_1 P - \alpha_3 P$.

To compute $kP$ for $P$ in the main subgroup of $E(\mathbb{F}_{2^m})$:

1. Compute $\alpha_u P$ for odd $u < 2^{w-1}$.
2. Compute $k' = k \bmod (\tau^m - 1)/(\tau - 1)$ in $\mathbb{Z}[\tau]$.
3. Find the width-$w$ $\tau$-adic expansion $\sum_{i=0}^{t} c_i \tau^i$ of $k'$, where $t \approx m$ and $c_i \in \{\pm\alpha_u\} \cup \{0\}$.
4. $Q \leftarrow \infty$.
5. For $i$ from $t$ to 0 do
   - 5.1 $Q \leftarrow \tau Q$.
   - 5.2 If $c_i \neq 0$ then add or subtract appropriate $\alpha_u P$.
6. Return $(Q)$.

No point doublings. Expect roughly $m/(w+1)$ point additions.

---

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

Point doubles and additions expected in finding $kP$ for a curve over $\mathbb{F}_{2^m}$ with $m = 233$.

| curve | method | doubles | adds |
|---|---|---|---|
| random | double and add | 232 | 116 |
| | NAF | 232 | 78 |
| | width-4 NAF | 1+232 | 3+47 |
| Koblitz | $\tau$-NAF | 0 | 78 |
| | width-4 $\tau$-NAF | 0 | 3+47 |

- Applying $\tau$ requires two or three field squarings, each costing roughly 15% of a field multiplication.
- Finding $k'$ (for the $\tau$-adic NAF) is not free.

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 $p \equiv 1 \pmod 3$. (In P-160, $p = 2^{160} - 229233$.)
- Let $E : y^2 = x^3 + b$, and let $\beta \in \mathbb{F}_p$ be an element of order 3.
- $\phi : (x,y) \mapsto (\beta x, y)$ is an endomorphism.
- Computing $\phi$ requires only 1 field multiplication.
- $|\phi| = 1$.

---

1. Gallant, Lambert, and Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms, CRYPTO 2001.
2. Park, Jeong, and Kim. An alternate decomposition of an integer for faster point multiplication on certain elliptic curves, PKC 2002.

# Using efficient endomorphisms...

Let $G \in E(\mathbb{F}_p)$ be a point of prime order $n$.

- $\phi$ acts on $\langle G \rangle$ by multiplication: $\phi P = \lambda P$, where $\lambda$ is a root (modulo $n$) of the characteristic polynomial of $\phi$. ($\lambda^2 + \lambda \equiv -1 \pmod{n}$ in the example.)
- To compute $kP$:
  - Write $k \equiv k_1 + k_2\lambda \pmod{n}$ where $|k_i| \approx \sqrt{n}$. (This can be done efficiently.)
  - $kP = k_1P + k_2\lambda P = k_1P + k_2\phi(P)$, which can be computed via interleaving.

| | | | | |
|---|---|---|---|---|
| $k_{1,t}$ | $\cdots$ | $k_{1,1}$ | $k_{1,0}$ | width-$w$ NAF of $k_1$ |
| $k_{2,t}$ | $\cdots$ | $k_{2,1}$ | $k_{2,0}$ | width-$w$ NAF of $k_2$ |

Approx half the doubles are eliminated. Cost of finding $k_i$ negligible if domain parameters are set in advance.

Solinas (CORR 2001-41) gives an example where finding $k_1$ and $k_2$ is free.

$$p = 2^{390} + 3 \equiv 1 \pmod 3$$
$$E : y^2 = x^3 - 2 \quad \text{over } \mathbb{F}_p$$
$$n = \#E(\mathbb{F}_p) = 2^{390} - 2^{195} + 7 = 63r, \quad r \text{ prime}$$
$$\lambda = \frac{2^{195}-2}{3},\ \beta = 2^{389} + 2^{194} + 1 \bmod p \implies \lambda(x,y) = (\beta x, y)$$

Write $k = k_2' 2^{195} + k_1'$ for $k_1' < 2^{195}$. Then

$$\begin{aligned} kP &= (2^{195}k_2' + k_1')P = ((3\lambda + 2)k_2' + k_1')P = \underbrace{(2k_2' + k_1')}_{k_1}P + \underbrace{3k_2'}_{k_2}\lambda P \\ &= k_1(x,y) + k_2((2^{389} + 2^{194} + 1)x, y) \end{aligned}$$

The cost of calculating $\beta x$ is less than a field multiplication.

# 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:

| | | |
|---|---|---|
| $x_2 = x^2 + b/x^2$ | | $x_2 = \lambda^2 + \lambda + a$ |
| $y_2 = x^2 + \lambda x_2 + x_2$ | or | $y_2 = x^2 + \lambda x_2 + x_2$ |
| (2 mul, 1 mul by $b$, 1 inv) | | (2 mul, 1 inv) |

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

$$x_2 = \lambda^2 + \lambda + a \quad \text{for } \lambda$$
$$y_2 = x^2 + \lambda x_2 + x_2 \quad \text{for } x$$

---

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. $\mathrm{Tr}(c) = c + c^2 + \cdots + c^{2^{m-1}} \in \{0, 1\}$.
2. The NIST random binary curves all have $\mathrm{Tr}(a) = 1$.
   $\mathrm{Tr}(x(kG)) = \mathrm{Tr}(a)$ 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
   $$\widehat{x}^2 = (\widehat{\lambda} + 1)x_2 + y_2$$
   $\mathrm{Tr}(x^2) = \mathrm{Tr}(x) = \mathrm{Tr}(a) = 1$, so $\mathrm{Tr}((\widehat{\lambda} + 1)x_2 + y_2)$ identifies $\lambda$.
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: $(x_2, \lambda_2)$ 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 $\widehat{\lambda}$. | $\approx 2/3$ field mult |
| 2. Find $T = x_2(\widehat{\lambda} + \lambda_2 + x_2 + 1)$ | $\approx$ 1 field mult |
| or $T = x_2(\widehat{\lambda} + 1) + y_2$ | |
| 3. If $\mathrm{Tr}(T) = 1$ then $\lambda = \widehat{\lambda}$, $x = \sqrt{T}$ | $\mathrm{Tr} \approx$ free |
| else $\lambda = \widehat{\lambda} + 1$, $x = \sqrt{T + x_2}$. | root $\approx 1/2$ field mult |
| 4. Return$(x, \lambda)$. | |

Conversion to affine $(x, \lambda) \to (x, y)$ is $\approx$ 1 field mult.

(Doubling in projective $\approx$ 4 field mults.)

# Calculating $kP$ by halve-and-add

**Algorithm** (halve-and-add, right-to-left)

INPUT: point $P$ and scalar $k$. OUTPUT: $kP$.

1. (Precomputation) Solve quadratic equations (21–30 field elements for B -163). Build table of 16 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. $Q \leftarrow \infty$.
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$(Q)$.

- Looks best when $I/M$ small.
- Multiple accumulators allow right-to-left with width-$w$ NAF variant.

## Summary: Efficient endomorphisms

1. If curve can be selected, GLV offers a dramatic speedup for on-line precomp case.

2. Halving is a significant improvement for $kP$ on binary curves in on-line precomp case, especially if $I/M$ is small.

3. Frobenius endomorphism $\tau$ on Koblitz curves is significantly better than halving.

4. Siet, Lange, Sica, Quisquater [SAC 2002] extend Koblitz-like expansions to other curves. Less useful if the endomorphism costs more than 1/2 point double.

5. Avanzi, Siet, and Sica [PKC 2004] give a scalar recoding that combines a halving step with Frobenius method on Koblitz curves. Trades some point additions for a halving.

   Practical interest may be limited, since it seems unlikely that there's room for halving code, but not for an additional point of storage and 3-TNAF.

**Example** $kP$ for the NIST random binary curve B-163 over field $\mathbb{F}_{2^{163}} = \mathbb{F}_2[z]/(z^{163}+z^7+z^6+z^3+1)$.

| Method | Coordinates | $w$ | Points stored | EC operations | | Field operations[a] | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | $A$ | $D$ | $M$ | $I$ | $I/M=5$ | $I/M=8$ |
| *Unknown point ($kP$, on-line precomputation)* | | | | | | | | | |
| Window NAF | affine | 4 | 3 | 35 | 163 | 396 | 198 | 1386 | 1980 |
| | projective | 4 | 3 | 3[b]+32 | 163 | 914 | 5 | 939 | 954 |
| Montgomery | affine | – | 0 | 162[c] | 162[d] | 328 | 325 | 1953 | 2928 |
| | projective | – | 0 | 162[c] | 162[d] | 982 | 1 | 987 | 990 |
| Halving $w$-NAF | affine | 5 | 7 | 7+27[e] | 1+163[f] | 423 | 35 | 598 | 705 |
| | projective | 4 | 3 | 6+30[e] | 3+163[f] | 671 | 2 | 681 | 687 |
| Window TNAF | affine | 5 | 7 | 34 | 0[g] | 114 | 34 | 284 | 386 |
| | projective | 5 | 7 | 7[b]+27 | 0[g] | 301 | 8 | 341 | 365 |

[a] Right columns give costs in terms of field mults for $I/M=5$ and $I/M=8$.
[b] Affine. [c] Addition for Montgomery. [d] $x$-coordinate only. [e] Cost $A+M$.
[f] Halvings; est. cost $2M$. [g] Field ops include applications of $\tau$ with $S=M/7$.

# Appendix: Timings (800 MHz Intel Pentium III)

| NIST curve | Method | Field mult M | Pentium III (800 MHz) normalized M | Pentium III (800 MHz) $\mu$s |
|---|---|---|---|---|
| *Unknown point ($kP$, on-line precomputation)* | | | | |
| P-192 | 5-NAF ($w = 5$) | 2016 | 2016 | 975 |
| B-163 | 4-NAF ($w = 4$) | 954 | 2953 | 1475 |
| B-163 | Halving ($w = 4$) | 687 | 2126 | 1050 |
| K-163 | 5-TNAF ($w = 5$) | 365 | 1130 | 625 |
| *Fixed base ($kP$, off-line precomputation)* | | | | |
| P-192 | Comb 2-table ($w = 4$) | 718 | 718 | 325 |
| B-163 | Comb | 386 | 1195 | 575 |
| K-163 | 6-TNAF ($w = 6$) | 263 | 814 | 475 |
| *Multiple point multiplication ($kP + lQ$)* | | | | |
| P-192 | Interleave ($w = 6, 5$) | 2306 | 2306 | 1150 |
| B-163 | Interleave ($w = 6, 4$) | 1154 | 3572 | 1800 |
| K-163 | Interleave TNAF ($w = 6, 5$) | 565 | 1749 | 1000 |

Timings using general-purpose registers. M is the estimated field multiplications with $I/M = 80$ and $I/M = 8$ in the prime and binary fields. Normalization gives equivalent P-192 field mults for this implementation.

1. Timings for Koblitz curves significantly faster than for random binary or prime in on-line precomp case.
2. Faster prime field multiplication gives P-192 the edge for off-line precomp case.
3. Results depend on processor and implementation.
   - Only general-purpose registers used.
   - Pentium III has floating-point registers which can accelerate prime field arithmetic, and single-instruction multiple-data (SIMD) registers that are easily harnessed for binary fields.
   - Integer multiplication with general-purpose registers on P-III is faster than on earlier or newer Pentium family processors. P-192 may be less competitive if hardware mult is weaker or operates on fewer bits.
4. The case where a large amount of storage is available for precomp in known-point methods is not addressed.

# Topic III

# ***Normal Basis Arithmetic***

# Normal bases in characteristic 2

Representing $\mathbb{F}_{2^m}$ field elements:

Polynomial basis: $\{1, x, \ldots, x^{m-1}\}$, reduction poly $f$.

Normal basis: $\{\beta^{2^0}, \beta^{2^1}, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$.

Motivation for use of normal basis reps:

- Squaring, square root are shifts. $x^2 + x = c$ can be solved bitwise.
- Performance of square root and quadratic solver is fundamental to methods based on point halving.
- Low complexity (Gaussian) NB bases have especially nice arithmetic.

Methods in 1990s seem to confirm that NB mult will be very slow in software compared to PB.

Ning & Yin [ICICS 2001] precompute shifts and significantly improve NB mult (at the cost of data-dependent storage).

Obtain the multplication table for basis $\{\beta^{2^i}\}$:

$$\beta^{2^i}\beta^{2^j} = \sum_{s=0}^{m-1} \lambda_{ij}^{(s)} \beta^{2^s},$$

Then $c = ab$ is given by

$$c_s = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_{i+s} b_{j+s} \lambda_{ij}^{(0)}.$$

Define $m \times m$ matrix $M = [\lambda_{ij}^{(0)}]$:

$$c_s = (a_s a_{s+1} \ldots a_{s+m-1}) M (b_s b_{s+1} \ldots b_{s+m-1})'.$$

- Number of 1s in $M$ is the *complexity* $C_M$.
- $C_M \geq 2m-1$. Basis is *optimal* if $C_M = 2m-1$.
- Optimal bases introduced by Mullin, Onyszchuk, Vanstone, and Wilson to reduce hardware complexity.

Optimal bases are relatively rare. Only $m = 233$ in the NIST recommended fields has an ONB.

Generalization: Gaussian normal bases.

- Let $p = mT + 1$ be a prime. $K = \langle u \rangle$ where $u \in \mathbb{Z}_p^*$ has order $T$.
- Suppose index $e$ of $\langle 2 \rangle$ in $\mathbb{Z}_p^*$ satisfies $\gcd(e, m) = 1$. Then
$$\mathbb{Z}_p^* = \{2^i u^j \mid 0 \le i < m,\ 0 \le j < T\},$$
and $K_i = K2^i$ for $0 \le i < m$ are the cosets of $K$ in $\mathbb{Z}_p^*$.
- $p \mid 2^{mT} - 1 \implies$ there is a primitive $p$th root of unity $\alpha \in \mathbb{F}_{2^{mT}}$.
- *Gauss periods* of type $(m, T)$ are $\beta_i = \sum_{j \in K_i} \alpha^j$ for $0 \le i < m$.

- Let $\beta = \beta_0$. Then $\beta_i = \beta^{2^i}$, and $\{\beta^{2^i}\}$ is a normal basis for $\mathbb{F}_{2^m}$ called a *type $T$ GNB*.
- $C_M \leq mT - 1$. $T$ is a measure of the complexity of the mult.
- ONBs are precisely the GNBs with $T \in \{1, 2\}$.

NIST recommended fields $\mathbb{F}_{2^m}$ and type of GNB.

| $m$ | 163 | 233 | 283 | 409 | 571 |
|---|---|---|---|---|---|
| Type | 4 | 2 | 6 | 4 | 10 |

Basic idea in Ning and Yin is precomputation of shifts.

- Let $\delta_i = \beta\beta^{2^i}$.
- Let $n_i$ be the number of 1s in NB rep of $\delta_i$, and let $w_{ik}$ satisfy $\delta_i = \sum_{k=1}^{n_i} \beta^{2^{w_{ik}}}$.
- Facts: $n_i \leq T$. $c = ab$ is given by

$$c_s = a_s b_{1+s} + \sum_{i=1}^{m-1} a_{i+s} \sum_{k=1}^{T} b_{w_{ik}+s}.$$

Rosing, Ning and Yin, and Reyhani-Masoleh and Hasan observed that the computation can be written:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} \odot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_0 \end{bmatrix} \oplus \sum_{i=1}^{m-1} \begin{bmatrix} a_i \\ a_{i+1} \\ \vdots \\ a_{i+m-1} \end{bmatrix} \odot \left( \begin{bmatrix} b_{w_{i1}} \\ b_{w_{i1}+1} \\ \vdots \\ b_{w_{i1}+m-1} \end{bmatrix} \oplus \cdots \oplus \begin{bmatrix} b_{w_{iT}} \\ b_{w_{iT}+1} \\ \vdots \\ b_{w_{iT}+m-1} \end{bmatrix} \right)$$

## Vector multiplication...

To find $c = ab$:

- Precompute all $m$ required shifts for $a$ and $b$. Copy the precomputation to simplify indexing in evaluation phase. Total storage: $4m$ words.
- Evaluation has $mT$ XOR and $m$ AND operations on field elements.

Variations (Dahab, H, Hu, Long, López, Menezes):

- Efficient one-table (e.g., precomputation for $a$ only) versions.
- For type 1, use matrix decompostion of Hasan, Wang, and Bhargava to reduce complexity to essentially $m$. (But type 1 means $m$ is even.)

**The good news**

- Significantly faster than preceding methods in software for NB.
- Easy to code and relatively easy to optimize.

**Some bad news**

- $2m$ or $4m$ words of dynamic storage.
- Still much slower than methods for polynomial basis.

$\mathbb{F}_{2^m}$ field multiplication (in $\mu$s), 800 MHz Intel Pentium III. Other than L-D, input and output are in NB.

| $m$ | Type | L-D comb | Ning&Yin 2 table | DHHLLM 1 table | DHHLLM Ring map |
|---|---|---|---|---|---|
| 162 | 1 | 1.3 | 6.7 | 5.0* | 2.7 |
| 163 | 4 | 1.3 | 9.6 | 8.4 | 10.4 |
| 233 | 2 | 2.3 | 11.4 | 11.7 | 7.1 |

*Uses matrix decomposition.

# Ring mapping method

Basic idea for GNB: map to an associated ring and use fast methods for polynomial basis reps.

*Basis conversion approach*

- Technique is well-known for the type 1 case, where the mapping is a permutation.
- Sunar and Koç [ToC 2001] is a basis conversion approach for type 2 that exploits properties of the map.

*Ring mapping approach*

- Arithmetic in the ring is modulo a cyclotomic polynomial (and so has especially simple reduction).
- “Palindromic representation” of Blake, Roth, and Seroussi is the special case for type 2.
- General case has a factor $T$ expansion, a significant hurdle.

Assume $\beta$ is a Gauss period of type $(m, T)$ and is a normal element. For $a \in \mathbb{F}_{2^m}$:

$$a = \sum_{i=0}^{m-1} a_i \beta^{2^i} = \sum_{i=0}^{m-1} a_i \sum_{j \in K_i} \alpha^j = \sum_{j=1}^{mT} a'_j \alpha^j$$

where $a'_j = a_i$ if $j \in K_i$.

- Let $R = \mathbb{F}_2[x]/(\Phi_p)$ where $\Phi_p(x) = (x^p - 1)/(x - 1)$.

  $$\phi : \sum_{i=0}^{m-1} a_i \beta^{2^i} \mapsto \sum_{j=1}^{mT} a'_j x^j$$

  is a ring homomorphism from $\mathbb{F}_{2^m}$ to $R$, and

  $\phi(\mathbb{F}_{2^m}) = \{c_1 x + \cdots + c_{mT} x^{mT} \in | \; c_j = c_k$ for $j, k \in K_i\}$.
- $\phi$ and its inverse are relatively inexpensive, and arithmetic in $R$ benefits from form of $\Phi_p$.

Naïve approach: map into the ring and then exploit fast polynomial-based arithmetic.

- There is a factor $T$ expansion, which can be significant. For $\mathbb{F}_{2^{163}}$, $T$ is at least 4.
- If $T$ is even (always the case if $m$ is odd), then the last $mT/2$ coefficients for elements in $\phi(\mathbb{F}_{2^m})$ are a mirror reflection of the first $mT/2$ [WHBG].
- Symmetry property is well-known in the case $T = 2$ where Gauss periods produce a type 2 optimal normal basis of the form
$$\{(\alpha + \alpha^{-1})^{2^i} \mid 0 \leq i < m\}$$
and there is an associated basis
$$\{\alpha^i + \alpha^{2m+1-i} \mid 1 \leq i \leq m\}.$$

# Gauss periods and mapping for small parameters

Consider $\mathbb{F}_{2^m}$ for $m = 3$. $T = 4$ gives prime $p = mT + 1 = 13$ and the unique subgroup of order $T = 4$ in $\mathbb{Z}_{13}^*$ is $K = \{1, -1, 5, -5\}$. The mapping into

$$R = \mathbb{F}_2[x]/(\Phi_p) \text{ for } \Phi_p(x) = (x^{13} - 1)/(x - 1)$$

is given by

$$\phi : a = (a_0, a_1, a_2) \mapsto (a_0, a_1, a_1, a_2, a_0, a_2, a_2, a_0, a_2, a_1, a_1, a_0).$$

- $\phi(a)$ is symmetric about $p/2$ and hence the first $(p-1)/2$ coeffs of the ring element suffice to invert $\phi$.
- Fewer coefficients may suffice: in the example, the first 4 (rather than $(p-1)/2 = 6$) coeffs will allow recovery of the field element.
- Wu, Hasan, Blake and Gao [ToC 2002] give sample minima for the number of consecutive $R$-element coeffs that permits recovery of the associated field element.

## Algorithm: Multiplication via ring mapping

INPUT: elements $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $b = \sum_{i=0}^{m-1} b_i \beta^{2^i}$ in $\mathbb{F}_{2^m}$.
OUTPUT: $c = ab = \sum_{i=0}^{m-1} c_i \beta^{2^i}$.

1. Calculate
$$a' = \phi(a) = \sum_{j=1}^{p-1} a'_j x^j$$
in $R$ where $a'_j = a_i$ if $j \in K_i$. Do similarly for $b'$.
2. Apply a fast multiplication method for polynomial-based reps to find half the coeffs of $c' = a'b'$ in $R$.
3. Return $c = \phi^{-1}(c')$.

# Observations on the ring mapping algorithm

1. $\phi$ can be optimized with per-field precomp. Each output coefficient is obtained with a word shift and mask. Only the first half are calculated in this fashion; remainder obtained by symmetry.
2. The López-Dahab "comb" method provides fast multiplication for polynomials and can be adapted to find only some of the output coefficients.
3. Reduction via
$$\Phi_p(x) = \frac{x^p - 1}{x - 1} = x^{p-1} + x^{p-2} + \cdots + x + 1$$
is fast.
4. Only half the output coefficients are required.
5. Cost of applying $\phi$ is approximately $T/2$ times the cost of $\phi^{-1}$.

# Example: ring mapping method for $m = 163$

$\mathbb{F}_{2^{163}}$ has a type $T = 4$ normal basis, and hence the mapping gives a factor 4 expansion.

- Modified comb mult finds $mT/2 = 326$ coeffs of the product $a'b'$. With 32-bit words, field elements require 6 words and ring elements require 21.
- Comb finds words $0, \ldots, 10, 19, \ldots, 30$ of the complete 42-word polynomial product.

  Small optimization: word 10 is not required since field elt can be recovered from $c'_1, \ldots, c'_{309}$ of product $c'$.
- The cost of an application of $\phi$ is approx 10% of the total field mult time ($\phi^{-1}$ costs approx half of $\phi$).

Experimentally, times on an Intel Pentium III are a factor 7 slower than field mult for a polynomial basis rep.

Competitive with the best methods with a NB rep.

# Example: ring mapping method for $m = 233$

$\mathbb{F}_{2^{233}}$ has a type $T = 2$ normal basis.

As expected, algorithm is faster for $m = 233$ (where 233 coefficients in the ring product are found) than for $m = 163$ (where 309 coefficients are found).

Method gives the fastest multiplication times for the type 2 case, and is approx a factor 3 slower than mult in a polynomial basis.

$\mathbb{F}_{2^m}$ field multiplication (in $\mu$s), 800 MHz Intel Pentium III. Other than L-D, input and output are in NB.

| $m$ | Type | L-D comb | Ning&Yin 2 table | DHHLLM 1 table | DHHLLM Ring map |
|---|---|---|---|---|---|
| 162 | 1 | 1.3 | 6.7 | 5.0* | 2.7 |
| 163 | 4 | 1.3 | 9.6 | 8.4 | 10.4 |
| 233 | 2 | 2.3 | 11.4 | 11.7 | 7.1 |

*Uses matrix decomposition.

# Memory consumption

Approximate code and storage requirements (in 32-bit words) for field multiplication in $\mathbb{F}_{2^m}$.

| | $m = 163$, $T = 4$ | | | $m = 233$, $T = 2$ | | |
|---|---|---|---|---|---|---|
| Storage type | Poly rep | Direct | Ring map | Poly rep | Direct | Ring map |
| object code & static data | 544 | 2092 | 4144 | 792 | 2740 | 3172 |
| automatic (stack) data | 108 | 360 | 360 | 146 | 510 | 260 |
| Total | 652 | 2452 | 4504 | 938 | 3250 | 3432 |

- Code for $\phi$ and $\phi^{-1}$ consume a significant portion of the total mem requirement. For type 2, however, total mem consumption is comparable to direct method.
- Algorithms for NB reps have significantly larger memory requirements than the comb method for PB.
- If total mem consumed by field arithmetic is the measurement of interest, then squaring, square root, and solving $x^2 + x = c$ for NB will likely have significantly smaller mem requirements than their counterparts for a PB.

# Summary: Normal basis arithmetic

For software implementations, Dahab, H, Hu, Long, López, and Menezes conclude:

1. Multiplication for normal basis reps significantly faster than previously reported.
2. Ring mapping method is competitive with best methods for low-complexity Gaussian normal bases (and superior for ONB).
3. Not sufficiently fast to help in two cases where NB are cited as especially desirable: Koblitz curves and point halving.

Penalty in NB mult appears to be sufficient in to overwhelm the advantages of fast and elegant operations of trace, squaring, square root, and solving $x^2 + x = c$.

# References: Normal basis arithmetic


1. I. F. Blake, R. M. Roth, and G. Seroussi. Efficient arithmetic in $GF(2^n)$ through palindromic representation. Technical Report HPL-98-134, Hewlett-Packard, Aug. 1998. Available via http://www.hpl.hp.com/techreports/.
2. R. Dahab, D. Hankerson, F. Hu, M. Long, J. López, and A. Menezes. Software multiplication using normal bases. Technical Report CACR 2004-12, University of Waterloo, 2004.
3. S. Gao, J. von zur Gathen, D. Panario, and V. Shoup. Algorithms for exponentiation in finite fields. *Journal of Symbolic Computation*, 29:879–889, 2000.
4. P. Ning and Y. Yin. Efficient software implementation for finite field multiplication in normal basis. *Information and Communications Security 2001*, LNCS 2229:177–189.

# References: Normal basis arithmetic...


5. A. Reyhani-Masoleh. Efficient algorithms and architectures for field multiplication using Gaussian normal bases. Technical Report CACR 2004-04, University of Waterloo, Canada, http://www.cacr.math.uwaterlo.ca, 2004.
6. B. Sunar and Ç. K. Koç. An efficient optimal normal basis type II multiplier. *IEEE Transactions on Computers*, 50(1):83–87, Jan. 2001.
7. H. Wu, A. Hasan, I. F. Blake, and S. Gao. Finite field multiplier using redundant representation. *IEEE Transactions on Computers*, 51(11):1306–1316, 2002.

# Topic IV

## *Inversion and affine arithmetic*

Motivation: optimizations for affine-coordinate arithmetic.

Example. Eisenträger, Lauter, and Montgomery [CT-RSA 2003] speed $2P+Q$ by omitting the $y$-coord in intermediate $P+Q$.

- Proposal is specific to affine coords, and will have significant number of inversions.
- Related papers by Ciet, Joye, Lauter and Montgomery, and Dimitrov, Imbert, and Mishra have similar requirement.

Problem: inversion seems to be expensive compared with multiplication.

- For the fastest multiplication times, [FHLM ToC 2004] have inversion cost $\approx$ 7–9 multiplications in $\mathbb{F}_{2^m}$.
- Inversion in prime fields significantly more expensive.

# *Inversion in $\mathbb{F}_{2^m}$ via EEA-like methods*

1. Euclidean algorithm.
   - Can efficently track size of (some) variables.
   - Requires explicit degree calculations.
   - Fastest in our tests on Pentium III (where *bit scan* instruction aids degree calc).

2. Binary Euclidean algorithm.
   - Can efficently track size of (some) variables.
   - No explicit degree calculations required.
   - Same cost for inversion as for division.
   - Slower in our tests on Pentium and SPARC family.

## Inversion in $\mathbb{F}_{2^m}$ via EEA-like methods

3. Almost inverse algorithm (2-stage method that first finds $a^{-1}z^k$ and then divides by $z^k$).

- Similar to BEA, but can track lengths of all variables.
- Tracking lengths efficiently seems to require code expansion.
- Variants allow fast 2nd stage even for non-favorable reduction poly.
- Fastest in our tests on SPARC (and competitive on Pentium).

# *Inversion in $\mathbb{F}_{2^m}$ via multiplication*

Inversion by multiplication uses

$$a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2.$$

- If $m$ is odd and $b = a^{2^{(m-1)/2}-1}$, then

  $$a^{2^{m-1}-1} = b \cdot b^{2^{(m-1)/2}}.$$

  Hence $a^{2^{m-1}-1}$ can be computed with one multiplication and $(m-1)/2$ squarings once $b$ has been evaluated.

- Recursive procedure finds $a^{-1}$ in

  $$\lfloor \log_2(m-1) \rfloor + w(m-1) - 1$$

  mults, where $w$ gives the number of 1s in binary rep.

For the NIST fields, this is 9–13 mults and $m-1$ squarings.

If squarings are free, speed is in ballpark of EEA-like methods. (But squarings are not free in our context.)

## Simultaneous Inversion

"Montgomery's Trick" to simultaneously find inverses is based on:

$$(x,y) \mapsto x \cdot y \mapsto \frac{1}{xy}, \quad x^{-1} = y \cdot \frac{1}{xy}, \quad y^{-1} = x \cdot \frac{1}{xy}$$

- Useful whenever an inverse costs more than 3 mults.
- Generalization: $k$ inverses have cost $I + 3(k-1)M$.

Example. For curves over prime fields, $3P + Q$ has cost $2I + 4S + 9M$ by simultaneous inversion for $2P$ and $P + Q$.

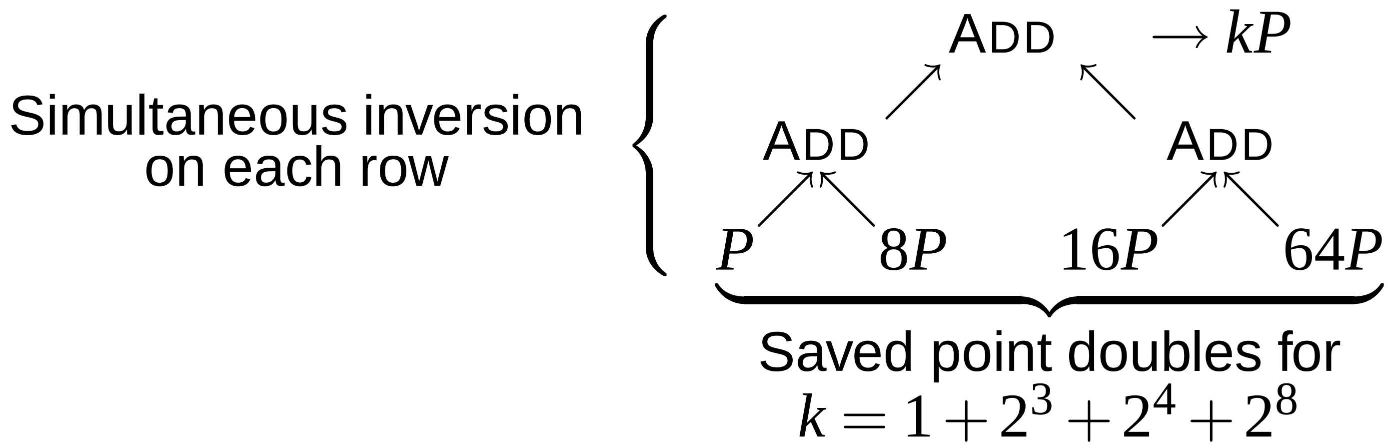
- $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $2P = (x_3, y_3)$, $P + Q = (x_4, y_4)$.
- 

| | | |
|---|---|---|
| $x_3 = \lambda_1^2 - 2x_1$ | $y_3 = (x_1 - x_3)\lambda_1 - y_1$ | $\lambda_1 = \frac{3x_1^2 + a}{2y_1}$ |
| $x_4 = \lambda_2^2 - x_1 - x_2$ | $y_4 = (x_1 - x_4)\lambda_2 - y_1$ | $\lambda_2 = \frac{y_1 - y_2}{x_1 - x_2}$ |

$\lambda_1$ and $\lambda_2$ are obtained with a single inversion.

The technique is applied widely. Efficiency (and storage) increases with the number simultaneous inverses.

Schroeppel and Beaver [2003] propose delaying point additions in $kP$ to exploit simultaneous inversion.




1. Calculate all point doubles, retaining those corresponding to required point additions.
2. Use binary tree to sum, with simultaneous inversion at each level. For a total of $t$ additions, have $\log_2 t$ inversions.

Attractive when point additions are a significant portion of the point mult. Examples: Koblitz curves and point halving.

In $\mathbb{F}_{2^m}$, cost of affine point additions reduced from $2M+I$ to approx $5M$. (Addition in mixed coords is approx $8M$.)

- First round additions more expensive if conversions required (e.g., in methods based on point halving).
- Additional storage is approx $m/3$ points (depends on rep used for $k$).
- Adapts to windowing via multiple accumulators.

Schroeppel and Beaver estimate 30% improvement for methods based on point halving in $\mathbb{F}_{2^{233}}$ (trinomial reduction poly). Uses estimate $H \approx 1.3M$.

Sanity test: suppose $H \approx 2M$, $I \approx 8M$, and NAF is used. Their cost estimate of $6M$/add gives 25% improvement.

Comparison of interest: in mixed coords, the additions cost $\approx (8+1)M$. Improvement is decreased to 20%.

Some side-channel information is eliminated if additions occur after all the point doubles.

Similar strategy has been applied in parallel processing; e.g. Mishra and Sarkar [PKC 2004].

# References


1. M. Ciet, M. Joye, K. Lauter and P. Montgomery. Trading Inversions for Multiplications in Elliptic Curve Cryptography. Designs, Codes and Cryptography. Cryptology ePrint Archive http://eprint.iacr.org/2003/257/.

2. V. Dimitrov, L. Imbert, and P. Mishra. Fast elliptic curve point multiplication using double-base chains. Cryptology ePrint Archive http://eprint.iacr.org/2005/069.

3. K. Eisenträger, K. Lauter, and P. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In *Topics in Cryptology—CT-RSA 2003* (LNCS 2612), 343–354, 2003.

4. R. Schroeppel and C. Beaver. Accelerating elliptic curve calculations with the reciprocal sharing trick. Mathematics of Public-Key Cryptography (MPKC), University of Illinois at Chicago, November 2003.

# Topic V

## ***Friendlier fields***

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

- $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/(f)$ for $p = 2^n \pm c$ and $f = x^m - \omega$.
- 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 register; $\mathbb{F}_{p^m}$ arithmetic can be performed via ops in $\mathbb{F}_p$.

**Attractions**

1. Arithmetic more elegant than in $\mathbb{F}_q$ for $q \approx p^m$.
2. Fast field inversion compared with $\mathbb{F}_q$.

**The bad news**

1. Inversion still expensive for small extensions.
2. Fastest mult looks like $\mathbb{F}_q$ case.

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. $A^{-1} = cA^{r-1}$.

**Cost**

- 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.

## Direct inversion for $m = 2$

Let $a = a_0 + a_1 x,\ f(x) = x^2 - \omega$, $\omega \in \mathbb{F}_p$. Want $a^{-1} = a'_0 + a'_1 x$.

- $aa^{-1} \equiv 1 \pmod f$ gives

$$\begin{pmatrix} a_0 & \omega a_1 \\ a_1 & a_0 \end{pmatrix} \begin{pmatrix} a'_0 \\ a'_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

- Then

$$a^{-1} = (a_0, -a_1)/\Delta, \quad \Delta = a_0^2 - \omega a_1^2.$$

Cost: $I + 2M + 2S$ (and a mult by $\omega$), all in the subfield.

## *Extending OEF: tower fields*

Idea: do a sequence of extensions with $f_i(x) = x^t - \omega_{i-1}$ irreducible over $GF(q^{t^{i-1}})$ and $f_i(\omega_i) = 0$.

- Main attraction: faster inversion than in OEF.
- OEF subfield inversion may require 40% of the total inversion time. Recursion in OTF $\Longrightarrow$ subfield inversion in $\mathbb{F}_q$ is expected to be less expensive.

**The bad news**

1. Multiplication slightly slower.
2. Unclear if inversion is fast enough to favor affine coords.
3. Analysis is limited. Parameters chosen (on ARM) may favor method.
4. Security of curves over OEFs and OTFs needs more analysis.

---

Baktir and Sunar [IEEE ToC 2004].

## Processor Adequate Finite Fields

Avanzi and Mihăilescu [SAC 2003] discuss PAFFs.

- Similar to OEF, but $p$ may be any prime subject to size restriction (size chosen to cooperate with hardware).
- Use “partial reduction” to reduce cost. Redundant rep.

Redundant reps and partial reduction is a widely-used technique.

1. Bernstein’s fast floating-point implementations for $\mathbb{F}_p$.
2. Fields with parameters not-of-special-form.
   - Gura, Eberle, and Chang Shantz, for $\mathbb{F}_{2^m}$.
   - Yanik, Savaş, and Koç, for $\mathbb{F}_p$.

Techniques have some application for special-form parameters.

Idea: move to a ring where where arithmetic is more pleasant.

Specialized case with fields having a type 1 ONB is well-known.

- Map field elts to $\mathbb{F}_2[x]/(f)$ where $f = \dfrac{x^{m+1}-1}{x-1}$.
- Do most arithmetic modulo $x^{m+1}-1$.
- Limitations: must have $m+1$ prime. Type 1 ONBs relatively rare.

Technique generalizes to Gaussian bases of type $T$, but with factor $T$ expansion.

# Move to a friendlier ring

Trinomials are desirable in reduction, solving $x^2 + x = c$ in point halving, and square root.

Doche gives “redundant trinomials” for $\mathbb{F}_{2^m}$ where there is no irreducible trinomial.

1. Find trinomial $x^n + x^k + 1$ that has an irreducible factor of degree $m$.

   Example: No irreducible trinomial for $m = 8$.
   Trinomial $x^{11} + x^5 + 1$ has irreducible factor $x^8 + x^6 + x^5 + x^4 + x^2 + x + 1$.

2. Do most arithmetic in the larger ring.
3. Expansion is $\leq$ 10 in 95% of cases. For 32-bit words, expansion does not require more words in 86% of cases.

4. *Optimal* redundant trinomials have degree divisible by 32.

   Example: $x^{197+27}+x^{103}+1$ has irreducible factor of degree 197.

   Faster, even though extension is usually larger. Optimal quadrinomials more common.

5. 20% improvement for reductions and squarings. Less than 5% for multiplications.

6. Inversion 15% slower.

Implementation was with NTL. More experimental data desirable.

Hyperelliptic curve of genus $g$ over $\mathbb{F}_q$:

$$v^2 + h(u)v = f(u)$$

$h, f \in \mathbb{F}_q[u]$, $\deg f = 2g+1$, $\deg h \leq g$.

- $g = 1 \implies$ elliptic curve.
- Known attacks $\implies g \leq 4$ (or $g \leq 3$) of interest.
- Arithmetic is in smaller fields for $g \in \{2, 3\}$, but curve operations are more complicated.
- Significant improvements in explicit formulae by Lange, Pezl, Wollinger, Guarjardo, Paar.

# Hyperelliptic vs elliptic curves

1. Avanzi [CHES 2004]: roughly 15% penalty with genus 2 curves over prime fields compared with EC.

   Limitations: “not interested in...prime moduli of special form” but this is a comparison of interest and may favor elliptic curves.

2. Lange and Stevens [SAC 2004]: genus 2 curves with $\deg h = 1$ are competitive or faster than EC for curves over binary fields.

   Limitations: implementation was with NTL. Affine coords only.

# References

1. R. Avanzi. Aspects of hyperelliptic curves over large prime fields in software implementations. CHES 2004, LNCS 3156:148–162.
2. Avanzi and P. Mihăilescu. Generic efficient arithmetic algorithms for PAFFs (Processor Adequate Finite Fields) and related algebraic structures. SAC 2003, LNCS 3006:320-334, 2004.
3. S. Baktir and B. Sunar. Optimal tower fields. IEEE Transactions on Computers 53:1231–1243, 2004.
4. C. Doche. Redundant trinomials for finite fields of characteristic 2. Cryptology ePrint Archive http://eprint.iacr.org/2004/055/.

5. T. Kobayashi, H. Morita, K. Kobayashi, F. Hoshino. Fast elliptic curve algorithm combining frobenius map and table reference to adapt to higher characteristic. EUROCRYPT '99, LNCS 1592:176-189. [Koblitz-like speedups. Downside: curve is over large subfield.]

6. V Müller. Efficient point multiplication for elliptic curves over special optimal extension fields. Public-Key Cryptography and Computational Number Theory, pages 197–207, de Gruyter, 2001. [Combines the ideas of GLV and OEF.]

7. T. Yanik, E. Savaş, and Ç. Koç. Incomplete reduction in modular arithmetic. IEE Proceedings: Computers and Digital Techniques, 149(2):46-52, March 2002.

# Topic VI

# *Using special-purpose hardware*

## Outline

1. Declining performance of integer arithmetic with general-purpose registers.
2. The floating-point approach.
3. “Multimedia” single-instruction multiple-data (SIMD) hardware.
4. Wish list: special instructions.

# Intel IA-32 processors

| Processor | Year | Selected features |
|---|---|---|
| 386 | 1985 | First IA-32 family processor with 32-bit operations and parallel stages. |
| 486 | 1989 | 5 pipelined stages in the 486; processor is capable of one instruction per clock cycle. |
| Pentium | 1993 | Dual-pipeline: optimal pairing gives two instructions per clock cycle. MMX added eight special-purpose 64-bit “multimedia” registers, supporting operations on vectors of 1, 2, 4, or 8-byte integers. |
| Pentium MMX | 1997 | |
| Pentium Pro | 1995 | P6 architecture has more sophisticated pipelining and out-of-order execution. Up to 3 $\mu$-ops executed per cycle. Improved branch prediction, but misprediction penalty much larger than on Pentium. Integer multiplication faster. SSE extensions on P-III have 128-bit registers supporting ops on vectors of single-precision floating-point values. |
| Pentium II | 1997 | |
| Celeron | 1998 | |
| Pentium III | 1999 | |
| Pentium 4 | 2000 | NetBurst architecture runs at significantly higher clock speeds, but many instructions have worse cycle counts than P6 family processors. SSE2 extensions have double-precision floating-point and 128-bit packed integer data types. |

### The good news

- Can do integer multiplication $32 \times 32 \rightarrow 64$ bits.
- Pentium II/III have faster multiplication than original Pentium and MMX.

### The bad news

- Must use $a$ and $d$ registers for the mult of interest.
- Multiplication on Pentium 4 with general-purpose registers is slower than on earlier processors.
- Pentium II/III have better branch prediction than the original Pentium, but mispredictions are more expensive.

# *Instruction latency/throughput*

Latency/throughput for Pentium II/III vs Pentium 4

| Instruction | Pentium II/III | Pentium 4 |
|---|---|---|
| Integer add, xor,... | 1 / 1 | .5 / .5 |
| Integer add, sub with carry | 1 / 1 | 6–8 / 2–3 |
| Integer multiplication | 4 / 1 | 14–18 / 3–5 |
| Floating-point multiply | 5 / 2 | 7 / 2 |
| MMX ALU | 1 / 1 | 2 / 2 |
| MMX multiply | 3 / 1 | 8 / 2 |

- Latency: number of clock cycles required before the result of an operation may be used.
- Throughput: number of cycles which must pass before the instruction may be executed again.

Small latency and small throughput are desirable.
Throughput can be significantly less than latency.

# ***Faster multiplication: floating-point***

Idea: use floating-point hardware to implement fast integer arithmetic.

- Potential for broad applicability: DEC Alpha, Intel Pentium, Sun SPARC, AMD Athlon.
- IEEE double-precision floating-point format

| $s$ | $e$ (11-bit exponent) | $f$ (52-bit fraction) |
|---|---|---|
| 63 | 62 … 52 | 51 … 0 |

  represents numbers $z = (-1)^s \times 2^{e-1023} \times 1.f$.
- Normalization of the significand $1.f$ increases effective precision to 53 bits.
- 80-bit double-extended format.
- Pentium has 8 floating-point registers. Length of the significand is selected in a control register.

## Some good news

- Floating-point addition operates on more bits than addition with integer instructions on 32-bit hardware.
- More registers and not restricted to specific registers. Can do useful things during the latency period.

## Some bad news

- Expensive to move between integer and floating-point formats.
- Bit operations which are convenient in integer format (e.g., extraction of specific bits, division by 2) are clumsy on values in floating point registers.
- Redundant rep $\Longrightarrow$ tests for equality are more expensive.

# Scalar multiplication for P-224

Bernstein: minimize conversions to/from floating point.

- P-224: NIST curve over $\mathbb{F}_p$ for prime $p = 2^{224} - 2^{96} + 1$.
- Performance improvements are in field arithmetic (and in the organization of field ops in point doubling and addition).
- Field multiplication will require more than 64 (floating-point) multiplications, compared with 49 in the classical method.
- On the positive side, more registers are available, mult can occur on any register, and products may be directly accumulated in a register without handling carry.

# P-224 field multiplication

| Multiplication in $\mathbb{F}_{p_{224}}$ | 1.7 GHz P4 |
|---|---|
| Classical integer | 0.62 |
| Karatsuba-Ofman | 0.82 |
| Floating-point | 0.20[a] |

[a]Excludes canonical form conversions.

**The good news**

1. Wide applicability. No coding in assembly.
2. Excluding conversions, can do $c = ab$ (to 8 FP values of roughly 28 bits) very fast.

**The bad news**

1. No assembly required, but have to manage scheduling and register allocation.
2. Can't allow compiler to unexpectedly spill 80-bit extended-double to 64-bit doubles. Alignment.
3. Conversion to canonical form is expensive, so must commit to FP across curve operations.
4. Algorithm verification.

Bernstein uses a width-4 window method (without sliding) for $kP$.

- Expected $3 + (15/16)(224/4)$ point additions.
- Scalar multiplication timings:

| | Cycles for $kP$ | |
| --- | --- | --- |
| Method | Pentium III | Pentium 4 |
| general-purpose registers | 1,200,000 | 2,700,000 |
| floating-point registers | 730,000 | 830,000 |

Reference implementation processes $k = \sum_{i=0}^{55} k_i 2^{4i}$ where $-8 \leq k_i < 8$. Precomp stores $iP$ in Chudnovsky coords $(X{:}Y{:}Z{:}Z^2{:}Z^3)$ for $i \in [-8, 8)$.

- Most of the improvement may be obtained by scheduling only field multiplication and squaring.
- Bernstein organized point arithmetic so that operations could be efficiently folded into field multiplication.
  - Point doubling $(x_2, y_2, z_2) = 2(x_1, y_1, z_1)$ is

$$\delta \leftarrow z_1^2, \quad \gamma \leftarrow y_1^2 \quad \beta \leftarrow x_1 \gamma, \quad \alpha \leftarrow 3(x_1 - \delta)(x_1 + \delta)$$
$$x_2 \leftarrow \alpha^2 - 8\beta, \quad z_2 \leftarrow (y_1 + z_1)^2 - \gamma - \delta, \quad y_2 \leftarrow \alpha(4\beta - x_2) - 8\gamma^2$$

  - 3 field mults, 5 squarings, 7 reductions.
- Expensive conversion to canonical form done only at the end of scalar multiplication.

# *Single-instruction multiple-data (SIMD)*

Perform operations in parallel on vectors. All Intel Pentiums except original and Pentium Pro.

- Initially "MMX Technology" for multimedia.

| Extension | Registers | Features added |
|---|---|---|
| MMX (PII) | 8 64-bit | vector ops on 1,2,4,8 byte integers; share space with floating-point registers |
| SSE (PIII) | 8 128-bit | vector ops on single-precision floating-point |
| SSE2 (P4) | 8 128-bit | vector ops on double-precision floating point and 64-bit integers |

- MMX suitable for binary field multiplication and inversion.
- SSE2 provides integer alternative to floating-point multiplication.

## *Single-instruction multiple-data (SIMD)*

- Advanced Micro Devices (AMD) K6 processor has MMX.
- Sun has Visual Instruction Set (VIS).

Basic idea applied to Intel and AMD processors:

- Implement fast 64-bit operations on primarily 32-bit machines.
- Gives more registers on register-poor machine.
- Integer multiplication (SSE2) can use 8 registers ($32 \times 32$ multiply with general-purpose registers has output in $a$ and $d$).
- SSE2 integer multiply has latency 8 and throughput 2. Can do useful things in the latency period.

Notation: integers $a = \sum a_i B^i$ for some $B = 2^w$ (e.g., $w = 28$ or $w = 32$). Want $c = ab$.

**1. Operand scanning approach.**

$$\begin{array}{c|ccccc}
 & a_0 & & a_1 & & a_2 \\
\hline
b_2 & a_0 b_2 & \cdots & a_1 b_2 & \cdots & a_2 b_2 & \longrightarrow & c + \sum a_i b_2 B^{i+2} \\
b_1 & a_0 b_1 & \cdots & a_1 b_1 & \cdots & a_2 b_1 & \longrightarrow & c + \sum a_i b_1 B^{i+1} \\
b_0 & a_0 b_0 & \cdots & a_1 b_0 & \cdots & a_2 b_0 & \longrightarrow & c + \sum a_i b_0 B^{i+0}
\end{array}$$

*Advantages*

- Control code is simple.
- Can use full 32-bit multiplications.

*Downsides*

- More memory accesses if few registers available.
- Must shift after each multiplication.

## 2. Product scanning method.

$$\begin{array}{c|ccccccc}
 & a_0 & & a_1 & & a_2 & & \\
\hline
b_2 & a_0b_2 & & a_1b_2 & & a_2b_2 & & \\
 & & \ddots & & \ddots & & \searrow & \\
b_1 & a_0b_1 & & a_1b_1 & & a_2b_1 & & \sum\limits_{i+j=4} a_ib_jB^4 \\
 & & \ddots & & \ddots & & \searrow & \\
b_0 & a_0b_0 & & a_1b_0 & & a_2b_0 & & \sum\limits_{i+j=3} a_ib_jB^3 \\
 & & \searrow & & \searrow & & \searrow & \\
 & & \sum\limits_{i+j=0} a_ib_jB^0 & & \sum\limits_{i+j=1} a_ib_jB^1 & & \sum\limits_{i+j=2} a_ib_jB^2 &
\end{array}$$

*Advantages*

- Possibly fewer memory accesses.
- Less shifting.

*Downsides*

- Control code more complicated (unless fully unrolled).
- To avoid carry, take $w < 32$. “Wastes” part of the multiplier.

1. GNU mp uses operand scanning.
2. Moore uses vector ops in 128-bit SSE2 to compute two products (with $w = 29$) simultaneously. Roughly operand scanning, but carry is handled in 2nd stage.
3. To avoid the requirement $w < 32$ in product scanning, use shuffle instruction to split 64-bit product $uv$:

$\longleftarrow$ 64 bits $\longrightarrow$ $\longleftarrow$ 64 bits $\longrightarrow$

| | | | |
|---|---|---|---|
| $0 \cdots 0$ | $0 \cdots 0$ | $uv_{\text{high}}$ | $uv_{\text{low}}$ |

$\downarrow$ *pshufd*

| | | | |
|---|---|---|---|
| $0 \cdots 0$ | $uv_{\text{high}}$ | $0 \cdots 0$ | $uv_{\text{low}}$ |

and then accumulate. Downside: have to shuffle on every mult.

Experiments suggest product scanning with scalar ops wins (even though input must be split and output reassembled).

# Multiplication with SSE2 integer ops

| Multiplication in $\mathbb{F}_{p_{224}}$ | Pentium 4 (1.7 GHz) |
|---|---|
| Classical integer (product scanning) | 0.62 |
| Karatsuba-Ofman (depth 2) | 0.82 |
| SIMD (SSE2; product scanning) | 0.27 |
| Floating-point | 0.20[a] |

[a]Excludes conversion to/from canonical form.

- Floating-point includes partial reduction to 8 floating-point values (each roughly 28 bits); does not include expensive conversion to canonical reduced form. Other times include reduction.
- Classical and Karatsuba would benefit from additional tuning specific to Pentium 4; regardless, both will be inferior to SIMD and floating-point.
- SIMD does not require the commitment of the floating-point approach.

## Summary: special-purpose hardware

1. Designs such as Pentium 4 and UltraSPARC have slow integer mult with general-purpose registers.
2. Common MMX subset suitable for binary field arithmetic. Relatively easy to code.
3. Floating-point hardware common on workstations speeds prime field arithmetic. Must commit to coding across curve operations.
4. Integer SSE2 extensions on Pentium 4 easy to insert locally, but not as fast as floating-point approach. Not available with Pentium II/III.

Wish list: Großschädl and Savaş [CHES 2004] propose instruction set extensions to speed field ops.

Amusements: use graphics card as a crypto co-processor [CT-RSA 2005].

1. D. Bernstein. A software implementation of NIST P-224. Presentation at the 5th Workshop on Elliptic Curve Cryptography (ECC 2001), University of Waterloo, October 29-31, 2001. Slides available from http://cr.yp.to/talks.html
2. J. Großschädl and E. Savaş. Instruction set extensions for fast arithmetic in finite fields $GF(p)$ and $GF(2^m)$. CHES 2004, LNCS 3156:133–147.
3. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.
4. S. Moore. Using Streaming SIMD Extensions (SSE2) to Perform Big Multiplications. Application Note AP-941, Intel Corporation, Version 2.0, Order Number 248606-001, 2000.