Efficient Software Implementation of Point Multiplication on Elliptic Curves

Kenny C.K. Fong

Centre for Applied Cryptographic Research
University of Waterloo

July 25, 2001

Motivation: ECDSA

- ECDSA stands for the **Elliptic Curve Digital Signature Algorithm** (Vanstone, 1992).
- ECDSA has been accepted as ISO, ANSI and IEEE standards.
- The security of ECDSA is based on the computational intractability of the *elliptic curve discrete logarithm problem* (ECDLP), which appears to be significantly harder than the discrete logarithm problem (used by DSA) and integer factorization problem (used by RSA).

Background: Finite Fields

- There are two kinds of finite fields (up to isomorphism):
  $\mathbb{F}_p = GF(p)$, where p is prime; and
  $\mathbb{F}_{p^m} = GF(p^m)$, where p is prime and $m > 1$ is an integer.
- $\mathbb{F}_p \cong \mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$
- $\mathbb{F}_{p^m} \cong \mathbb{Z}_p[x]/(f(x)) = \{a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1} : a_0, \ldots, a_{m-1} \in \mathbb{Z}_p\}$, where $f(x)$ is an irreducible polynomial over $\mathbb{Z}_p$ of degree m

Background: Elliptic Curves

- An elliptic curve E over $\mathbb{F}_p$ ($p > 3$) is defined by an equation of the form

$$y^2 = x^3 + ax + b \tag{1}$$

where $a, b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

- The set $E(\mathbb{F}_p)$ consists of all points (x, y), $x, y \in \mathbb{F}_p$, which satisfy equation (1), together with a special point $\mathcal{O}$ called the *point at infinity*.

- $E(\mathbb{F}_p)$ forms an (additive) **abelian group**.

Basic Curve Operations

Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_p)$.

- (Point negation) $-P = (x_1, -y_1)$.
- (Point addition) Assume $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

- (Point doubling) Assume $P \neq -P$. Then $2P = (x_4, y_4)$, where

$$x_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \quad y_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

ECDSA: Domain Parameters

- a finite field $\mathbb{F}_q = GF(q)$ of order q
- an elliptic curve E over $\mathbb{F}_q$
- a *base point* $G \in E(\mathbb{F}_q)$ of prime order n, where $n > 2^{160}$ and $n > 4\sqrt{q}$ ($< G >= \{kG : k \in \mathbb{Z}\}$ is a cyclic subgroup of $E(\mathbb{F}_q)$ of order n)

Most standards restrict the underlying finite field $\mathbb{F}_q$ to be a *prime field* ($q = p$ where $p > 3$ is prime) or a *binary field* ($q = 2^m$ where $m > 1$ is an integer).

ECDSA: Signature Generation

Let (d, Q) be the key pair, where the private key $d \in [1, n)$ is a (pseudo)random integer pre-selected and $Q = dG$ is the public key. Let m be the message to be signed.

1. Select a (pseudo)random integer $k \in [1, n)$.
2. Compute $\mathbf{kG} = (x_1, y_1)$ and convert x_1 to an integer $\bar{x}_1$.
3. Compute $r = \bar{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4. Compute $k^{-1} \bmod n$.
5. Compute SHA-1(m) and convert this bit string to an integer e.
6. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
7. The signature for the message m is (r, s).

ECDSA: Signature Verification

1. Verify that r and s are integers in the interval $[1, n)$.
2. Compute SHA-1(m) and convert this bit string to an integer e.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = (x_2, y_2) = \mathbf{u_1 G} + \mathbf{u_2 Q}$.
6. If $X = \mathcal{O}$, then reject the signature. Otherwise, convert x_2 to an integer $\bar{x}_2$, and compute $v = \bar{x}_2 \bmod n$.
7. Accept the signature if and only if $v = r$.

Prime Field Operations: Reduction

Let $\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$ be a prime field where p is an n-bit prime, $n \geq 160$ (i.e. $p \geq 2^{159}$). Let c be an integer where $0 \leq c \leq (p-1)^2$ (so c can be $2n$-bit long). Find a number $r \in \mathbb{F}_p$ such that $c \equiv r \pmod{p}$.

- Barrett reduction and Montgomery reduction
- **Mersenne primes** or **Mersenne-like primes**

  e.g. Mersenne prime $p = 2^{521} - 1$, $n = 521$

  $2^{521} \equiv 1 \pmod{p}$

  $c = c_1 2^{521} + c_0 \equiv c_0 + c_1 \pmod{p}$

  (c_0 and c_1 are 521-bit long)

Fast Reduction: Example

Consider the NIST-recommended prime $p_{192} = 2^{192} - 2^{64} - 1$.

INPUT: integer $c = (c_5, c_4, c_3, c_2, c_1, c_0) = \sum_{i=0}^{5} c_i 2^{64i}$ where each c_i is a 64-bit word, and $0 \le c \le (p_{192} - 1)^2$.
OUTPUT: $c \bmod p_{192}$.

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 $(s_1 + s_2 + s_3 + s_4) \bmod p_{192}$.

Fast Reduction: Arithmetic

Note: $p_{192} = 2^{192} - 2^{64} - 1 \implies 2^{192} \equiv 2^{64} + 1 \pmod{p_{192}}$

$$
\begin{aligned}
c &= (c_5, c_4, c_3, c_2, c_1, c_0) \\
&= (c_5, c_4, c_3)2^{192} + (c_2, c_1, c_0) \\
&\equiv (c_5 2^{128} + c_4 2^{64} + c_3)(2^{64} + 1) + s_1 \\
&\equiv c_5 2^{128}(2^{64} + 1) + c_4 2^{64}(2^{64} + 1) + c_3(2^{64} + 1) + s_1 \\
&\equiv c_5(2^{192} + 2^{128}) + c_4(2^{128} + 2^{64}) + c_3(2^{64} + 1) + s_1 \\
&\equiv c_5(2^{128} + 2^{64} + 1) + (c_4, c_4, 0) + (0, c_3, c_3) + s_1 \\
&\equiv (c_5, c_5, c_5) + s_3 + s_2 + s_1 \\
&\equiv s_1 + s_2 + s_3 + s_4 \pmod{p_{192}}
\end{aligned}
$$

Point Multiplication: Binary Method

INPUT: $k = (k_{m-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_p)$.
OUTPUT: kP.

1. $Q \longleftarrow \mathcal{O}$.
2. For i from $m-1$ downto 0 do
   - $Q \longleftarrow 2Q$.
   - If $k_i = 1$ then $Q \longleftarrow Q + P$.
3. Return Q.

e.g. $k = 13 = 1101_2$

$$13P = P + 2(\mathcal{O} + 2(P + 2(P + \mathcal{O})))$$

$$Q : \mathcal{O} \to P \to 2P \to 3P \to 6P \to 12P \to 13P$$

Point Multiplication: Gallant Method

Suppose that $P \in E(\mathbb{F}_p)$ is not known a priori.

- The larger the value of k, the longer it takes to calculate kP.
- A new technique (Gallant, Lambert, Vanstone, 2000) has been proposed which speeds up point multiplication of elliptic curves having an **efficiently-computable endomorphism**.
- The approach calculates kP by decomposing k into two *smaller* scalars k_1 and k_2, and applies just one application of the endomorphism.
- A speedup of up to 50% is expected over the best general methods for point multiplication.

Endomorphisms

Let E be an elliptic curve defined over $\mathbb{F}_q$.

- An *endomorphism* of E defined over $\mathbb{F}_q$ is a map $\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_q)$ satisfying $\phi(\mathcal{O}) = \mathcal{O}$.
- ϕ is a group homomorphism of $E(\mathbb{F}_q)$.
- *Example.* For each $m \in \mathbb{Z}$, the *multiplication by* m map $[m] : E(\mathbb{F}_q) \to E(\mathbb{F}_q)$ defined by $P \mapsto mP$ is an endomorphism defined over $\mathbb{F}_q$.

Point Multiplication and Endomorphism

Suppose that the point P has prime order n, and the characteristic polynomial of ϕ has a root λ modulo n. Then $\phi(Q) = \lambda Q$ for all $Q \in < P >$.

Example. Consider the elliptic curve

$$E : y^2 = x^3 + b \tag{2}$$

defined over $\mathbb{F}_p$, where $p \equiv 1 \pmod 3$ is a prime.

Efficient Endomorphism: Example

- $\mathbb{F}_p$ has an element β of order 3 ($\beta^3 \equiv 1 \pmod{p}$).
- The map $\phi : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$ defined by $(x, y) \mapsto (\beta x, y)$ and $\mathcal{O} \mapsto \mathcal{O}$ is an endomorphism defined over $\mathbb{F}_p$.
- Suppose that $P \in E(\mathbb{F}_p)$ is a point of prime order n.
- There exists an integer λ satisfying $\lambda^2 + \lambda + 1 \equiv 0 \pmod{n}$.
- $\phi(Q) = \lambda Q$ for all $Q \in < P >$.
- $\phi(Q)$ can be computed using only one field multiplication.

Using the Endomorphism

- Since n is the order of the point P, $nP = \mathcal{O}$.
- Suppose that we can *efficiently* write $k \equiv k_1 + k_2\lambda \pmod{n}$, where $k_1, k_2 \in [0, \lfloor\sqrt{n}\rfloor]$. Then $k = k_1 + k_2\lambda + nq$ for some $q \in \mathbb{Z}$.
- It follows that

$$\begin{aligned} kP &= (k_1 + k_2\lambda + nq)P \\ &= k_1P + k_2\lambda P + nqP \\ &= k_1P + k_2(\lambda P) + q(nP) \\ &= k_1P + k_2\phi(P) + \mathcal{O} \\ &= k_1P + k_2\phi(P). \end{aligned}$$

Point Multiplication: Interleaving Method

INPUT: $u = (u_{m-1}, \ldots, u_0)_2$, $v = (v_{m-1}, \ldots, v_0)_2$, $P \in E(\mathbb{F}_p)$.
OUTPUT: kP.

1. $Q \longleftarrow \phi(P)$.
2. $A \longleftarrow \mathcal{O}$.
3. For i from $m-1$ downto 0 do
   - $A \longleftarrow 2A$.
   - If $u_i = 1$ then $A \longleftarrow A + P$.
   - If $v_i = 1$ then $A \longleftarrow A + Q$.
4. Return A.

Interleaving Method: Analysis

- Suppose that k is t-bit long.
- $k_1, k_2 \in [0, \lfloor \sqrt{n} \rfloor]$ means that k_1 and k_2 are $(t/2)$-bit long, i.e. k_1 and k_2 are *small*.
- The interleaving method processes the i-th bit of both k_1 and k_2 together in the i-th iteration, hence it saves half of the point doublings.

Decomposing k: Formulation

- Consider the homomorphism $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}_n$ defined by $(i, j) \mapsto (i + \lambda j) \bmod n$.
- Let $k \in [1, n)$. We wish to find *small* integers $k_1, k_2 \in [0, \lfloor \sqrt{n} \rfloor]$ such that $k \equiv k_1 + k_2\lambda \pmod{n}$.
- Equivalently, we wish to find a vector $\vec{v} = (k_1, k_2) \in \mathbb{Z} \times \mathbb{Z}$ of *small* Euclidean norm such that $f(\vec{v}) = k$.
- Clearly, $f((k, 0)) = k$.

Decomposing k: Algorithm

Precomputation:
Find linearly independent vectors $\vec{v}_1, \vec{v}_2 \in \mathbb{Z} \times \mathbb{Z}$ of *small* Euclidean norm such that $f(\vec{v}_1) = f(\vec{v}_2) = 0$.

1. Find a vector $\vec{u} = l_1\vec{v}_1 + l_2\vec{v}_2$ ($l_1, l_2 \in \mathbb{Z}$) in the integer lattice generated by $\vec{v}_1$ and $\vec{v}_2$ that is *close* to $(k, 0)$.
   - Write $(k, 0) = \beta_1\vec{v}_1 + \beta_2\vec{v}_2$, $\beta_1, \beta_2 \in \mathbb{Q}$.
   - Set $\vec{u} = l_1\vec{v}_1 + l_2\vec{v}_2$, where $l_1 = \lfloor \beta_1 \rceil$ and $l_2 = \lfloor \beta_2 \rceil$.
2. Set $\vec{v} = (k_1, k_2) = (k, 0) - \vec{u}$.

Decomposing k: Analysis

- Notice that

$$
\begin{aligned}
f(\vec{v}) &= f((k,0) - \vec{u}) \\
&= f((k,0)) - f(\vec{u}) \\
&= k - f(l_1\vec{v}_1 + l_2\vec{v}_2) \\
&= k - l_1 f(\vec{v}_1) - l_2 f(\vec{v}_2) \\
&= k - l_1(0) - l_2(0) \\
&= k.
\end{aligned}
$$

- The vector $\vec{v}$ is indeed short. One can show that $\|\vec{v}\| \leq max\{\|\vec{v}_1\|, \|\vec{v}_2\|\}$.

Decomposing k: Precomputation

- The two linearly independent short vectors $\vec{v}_1$ and $\vec{v}_2$, where $f(\vec{v}_1) = f(\vec{v}_2) = 0$, can be found using the *Extended Euclidean Algorithm.*

$$
\begin{aligned}
s_0\lambda + t_0 n &= r_0 \\
s_1\lambda + t_1 n &= r_1 \\
&\vdots \\
s_i\lambda + t_i n &= r_i \approx \sqrt{n} \\
&\vdots \\
s_l\lambda + t_l n &= 1
\end{aligned}
$$

Decomposing k: Technicalities

Denote $A = [\vec{v}_1|\vec{v}_2]$.

$$
\begin{aligned}
(k, 0) &= \beta_1 \vec{v}_1 + \beta_2 \vec{v}_2 \\
A \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} &= \begin{bmatrix} k \\ 0 \end{bmatrix} \\
\begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} &= A^{-1} \begin{bmatrix} k \\ 0 \end{bmatrix}
\end{aligned}
$$