Elliptic Curve Point Multiplication using Halving

Darrel Hankerson (Auburn University)
Alfred Menezes (University of Waterloo)

May 11, 2004

Elliptic curve cryptographic schemes require calculations of the type

$$kP = \underbrace{P + \cdots + P}_{k}$$

where k is a large integer and the addition is over the elliptic curve (see elliptic curves). The operation is known as *scalar* or *point multiplication*, and dominates the execution time of signature and encryption schemes based on elliptic curves. Double-and-add variations of familiar square-and-multiply methods (see square and multiply algorithm) for modular exponentiation are commonly used to find kP. Windowing methods can significantly reduce the number of point additions required, but the number of point doubles remains essentially unchanged.

Among techniques to reduce the cost of the point doubles in point multiplication, perhaps the best known is illustrated in the case of Koblitz curves (elliptic curves over $\mathbb{F}_{2^m}$ with coefficients in $\mathbb{F}_2$; see [8]), where point doubling is replaced by inexpensive field squarings. Knudsen [4] and Schroeppel [6, 7] proposed a point halving operation which shares strategy with τ-adic methods on Koblitz curves in the sense that most point doublings are replaced with less-expensive operations. The improvement is not as dramatic as that obtained on Koblitz curves; however, halving applies to a wider class of curves.

We restrict our attention to elliptic curves E over binary fields $\mathbb{F}_{2^m}$ defined by the equation

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

where $a, b \in \mathbb{F}_{2^m}$, $b \neq 0$. To simplify the exposition, we consider only the case that $\mathrm{Tr}(a) = 1$; see [4] (where "minimal two-torsion" corresponds to $\mathrm{Tr}(a) = 1$) for the necessary adjustments and computational costs for $\mathrm{Tr}(a) = 0$ curves. We further assume that m is prime. These properties are satisfied by the five random curves over binary fields recommended by NIST in the FIPS 186-2 standard [2].

Let $P = (x, y)$ be a point on E with $P \neq -P$. The (affine) coordinates of $Q = 2P = (u, v)$ can be computed as follows:

$$\lambda = x + y/x \tag{1}$$
$$u = \lambda^2 + \lambda + a \tag{2}$$
$$v = x^2 + u(\lambda + 1) \tag{3}$$

(see elliptic curves). *Point halving* is the following operation: given $Q = (u, v)$, compute $P = (x, y)$ such that $Q = 2P$. The basic idea for halving is to solve (2) for λ, (3) for x, and finally (1) for y.

When $\mathcal{G}$ is a subgroup of odd order n in E, point doubling and point halving are automorphisms of $\mathcal{G}$. Therefore, given a point $Q \in \mathcal{G}$, there is a unique point $P \in \mathcal{G}$ such that $Q = 2P$. An efficient algorithm for point halving in $\mathcal{G}$, along with a point multiplication algorithm based on halving, are outlined in the following sections.

1 Point halving

The notion of *trace* plays a central role in deriving an efficient algorithm for point halving. The *trace function* $\mathrm{Tr} : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ is defined by $\mathrm{Tr}(c) = c + c^2 + c^{2^2} + \cdots + c^{2^{m-1}}$. The map is linear, $\mathrm{Tr}(c) \in \{0, 1\}$, and $\mathrm{Tr}(u) = \mathrm{Tr}(a)$ for $(u, v) \in \mathcal{G}$ (from an application of Tr to (2)).

Given $Q = (u, v) \in \mathcal{G}$, point halving seeks the unique point $P = (x, y) \in \mathcal{G}$ such that $Q = 2P$. The first step of halving is to find $\lambda = x + y/x$ by solving the equation

$$\widehat{\lambda}^2 + \widehat{\lambda} = u + a \tag{4}$$

for $\widehat{\lambda}$. It is easily verified that $\lambda \in \{\widehat{\lambda}, \widehat{\lambda} + 1\}$. If $\mathrm{Tr}(a) = 1$, then it follows from (3) that $\widehat{\lambda} = \lambda$ if and only if $\mathrm{Tr}(v + u\widehat{\lambda}) = 0$. Hence λ can be identified, and then (3) is solved for the unique root x. Finally, if needed, $y = \lambda x + x^2$ may be recovered with one field multiplication.

Let the λ-*representation* of a point $Q = (u, v)$ be (u, λ_Q), where $\lambda_Q = u + v/u$. Given the λ-representation of Q as the input to point halving, we may compute $t = v + u\widehat{\lambda}$ without converting to affine coordinates, since

$$t = v + u\widehat{\lambda} = u\left(u + u + \frac{v}{u}\right) + u\widehat{\lambda} = u(u + \lambda_Q + \widehat{\lambda}).$$

In point multiplication, repeated halvings may be performed directly on the λ-representation of a point, with conversion to affine only when a point addition is required.

Algorithm 1 Point halving

INPUT: λ-representation (u, λ_Q) or affine representation (u, v) of $Q \in \mathcal{G}$.

OUTPUT: λ-representation (x, λ_P) of $P = (x, y) \in \mathcal{G}$, where $\lambda_P = x + y/x$ and $Q = 2P$.

1. Find a solution $\widehat{\lambda}$ of $\widehat{\lambda}^2 + \widehat{\lambda} = u + a$.
2. If the input is in λ-representation, then compute $t = u(u + \lambda_Q + \widehat{\lambda})$;
   else compute $t = v + u\widehat{\lambda}$.
3. If $\mathrm{Tr}(t) = 0$, then $\lambda_P \leftarrow \widehat{\lambda}$, $x \leftarrow \sqrt{t + u}$;
   else $\lambda_P \leftarrow \widehat{\lambda} + 1$, $x \leftarrow \sqrt{t}$.
4. Return (x, λ_P).

The point halving algorithm requires a field multiplication and three main steps: computing the trace of t, solving the quadratic equation (4), and computing a square root. In a normal basis, field elements are represented in terms of a basis of the form $\{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\}$. The trace of an element $c = \sum c_i \beta^{2^i} = (c_{m-1}, \ldots, c_0)$ is given by $\mathrm{Tr}(c) = \sum c_i$. The square root computation is a right rotation: $\sqrt{c} = (c_0, c_{m-1}, \ldots, c_1)$. Squaring is a left rotation, and $x^2 + x = c$ can be solved bitwise. These operations are expected to be inexpensive relative to field multiplication. However, field multiplication in software for normal basis representations tends to be slow in comparison to multiplication with a polynomial basis. We shall restrict our discussion to computations in a polynomial basis representation, where $c \in \mathbb{F}_{2^m}$ is expressed as $c = \sum_{i=0}^{m-1} c_i z^i$ with $c_i \in \{0, 1\}$.

Trace computations The trace of c may be calculated as $\mathrm{Tr}(c) = \sum_{i=0}^{m-1} c_i \, \mathrm{Tr}(z^i)$, where the values $\mathrm{Tr}(z^i)$ are precomputed. As an example, $\mathbb{F}_{2^{163}}$ with reduction polynomial $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$ has $\mathrm{Tr}(z^i) = 1$ if and only if $i \in \{0, 157\}$, and finding $\mathrm{Tr}(c)$ is an essentially free operation.

Solving the quadratic equation For an odd integer m, define the *half-trace* $H : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ by $H(c) = \sum_{i=0}^{(m-1)/2} c^{2^{2i}}$. Then

$$H(c) = H\big(\sum_{i=0}^{m-1} c_i z^i\big) = \sum_{i=0}^{m-1} c_i H(z^i)$$

is a solution of the equation $x^2 + x = c + \mathrm{Tr}(c)$. For elements c with $\mathrm{Tr}(c) = 0$, the calculation produces a solution $H(c)$ of $x^2 + x = c$, and requires an expected $m/2$ field additions and storage for m field elements $H(z^i)$.

The storage and time required to solve the quadratic equation can be reduced. The basic strategy is to write $H(c) = H(c') + s$ where c' has fewer nonzero coefficients than c.

The property $H(c) = H(c^2) + c + \mathrm{Tr}(c)$ for $c \in \mathbb{F}_{2^m}$ may be applied directly to eliminate storage of $H(z^i)$ for even i. Repeated applications may yield further improvements. For example, if the reduction polynomial $f(z) = z^m + r(z)$ has $\deg r < m/2$, then the strategy can be applied in an especially straightforward fashion to eliminate storage of $H(z^i)$ for odd i, $m/2 < i < m - \deg r$. If $\deg r$ is small, the storage requirement is reduced to approximately $m/4$ elements. Such strategies are outlined in [4, Appendix A]; see also [3] for details.

Computing square roots The square root of c may be expressed as

$$\sqrt{c} = \sum_{i \text{ even}} c_i z^{\frac{i}{2}} + \sqrt{z} \sum_{i \text{ odd}} c_i z^{\frac{i-1}{2}}.$$

The value $\sqrt{z}$ may be precomputed, and finding $\sqrt{c}$ is expected to be significantly less expensive than a field multiplication. Note that if the reduction polynomial f is a trinomial, then substantial improvements are possible based on the observation that $\sqrt{z}$ may be obtained directly from f; see [3].

2 Point multiplication

Halve-and-add variants of point multiplication methods replace most point doublings with halvings. However, point halving is performed on affine (or λ) representations, and hence some modifications may be required if projective coordinates are used. The algorithm presented in this section illustrates the use of projective coordinates (and a windowing method) with halving.

Let $w \geq 2$ be an integer. A *width-w NAF* of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where each nonzero coefficient k_i is odd, $|k_i| < 2^{w-1}$, $k_{l-1} \neq 0$, and at most one of any w consecutive digits is nonzero [8]. A positive integer k has a unique width-w NAF denoted $\mathrm{NAF}_w(k)$, with length at most one more than the length of the binary representation. As an illustration, coefficients in the binary representation of $k = 29$ and the width-2 and width-3 NAFs are given by:

$$\begin{array}{rccccc} k = & 1 & 1 & 1 & 0 & 1 \\ \mathrm{NAF}_2(k) = 1 & 0 & 0 & -1 & 0 & 1 \\ \mathrm{NAF}_3(k) = 1 & 0 & 0 & 0 & 0 & -3 \end{array}$$

The average density of nonzero digits among all width-w NAFs of length l is approximately $1/(w+1)$. The signed digit representation $\mathrm{NAF}_2(k)$ is known as the non-adjacent form. In point multiplication, the use of signed digit representations is motivated by the property that point subtraction is as efficient as addition

If kP is to be found for a given scalar k, then a conversion is required for halving-based methods. If k' is defined by

$$k \equiv k'_{t-1}/2^{t-1} + \cdots + k'_2/2^2 + k'_1/2 + k'_0 \pmod{n}$$

where $k'_i \in \{0, 1\}$ and n is the order of $\mathcal{G}$, then $kP = \sum_{i=0}^{t-1} k'_i/2^i P$; i.e., $(k'_{t-1}, \ldots, k'_0)$ may be used by halving-based methods. This can be generalized to width-w NAF: if $\sum_{i=0}^{l-1} k'_i 2^i$ is the w-NAF representation of $2^{t-1}k \bmod n$, then

$$k \equiv \sum_{i=0}^{t-1} \frac{k'_{t-1-i}}{2^i} + 2k'_t \pmod{n}$$

where it is understood that $k_i = 0$ if $i \geq l$.

Algorithm 2 presents a right-to-left version of a halve-and-add method with the input $2^{t-1}k \bmod n$ represented in w-NAF. Point halving occurs on the input P rather than on accumulators (which may be in projective form). The expected running time is approximately

$$(\text{step 4 cost}) + (t/(w+1) - 2^{w-2})A' + tH$$

where H denotes a point halving and A' is the cost of a point addition when one of the inputs is in λ-representation. If projective coordinates are used for Q_i, then the additions in step 3.1 and 3.2 are mixed-coordinate. Step 4 may be performed by calculating $Q_i \leftarrow Q_i + Q_{i+2}$ for odd i from $2^{w-1}-3$ to 1, and then the result is given by $Q_1 + 2\sum_{i \in I \setminus \{1\}} Q_i$ [5, Exercise 4.6.3-9].

Algorithm 2 Halve-and-add w-NAF (right-to-left) point multiplication

INPUT: Window width w, $\text{NAF}_w(2^{t-1}k \bmod n) = \sum_{i=0}^{t} k'_i 2^i$, $P \in \mathcal{G}$.

OUTPUT: kP. (Note: $k = k'_0/2^{t-1} + \cdots + k'_{t-2}/2 + k'_{t-1} + 2k'_t \mod n$.)

1. Set $Q_i \leftarrow \infty$ for $i \in I = \{1, 3, \ldots, 2^{w-1} - 1\}$.
2. If $k'_t = 1$ then $Q_1 = 2P$.
3. For i from $t-1$ downto 0 do:
   - 3.1 If $k'_i > 0$ then $Q_{k'_i} \leftarrow Q_{k'_i} + P$.
   - 3.2 If $k'_i < 0$ then $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$.
   - 3.3 $P \leftarrow P/2$.
4. $Q \leftarrow \sum_{i \in I} iQ_i$.
5. Return(Q).

The computational costs in terms of field operations are summarized in Table 1 for the case that Algorithm 2 is used with $w = 2$. Only field multiplications and inversions are considered, under the assumption that field addition is relatively inexpensive. The choice between affine coordinates and projective coordinates is driven primarily by the cost of inversion (I) relative to multiplication (M). If division has approximate cost $I + M$, then the estimates in the table show that projective coordinates will be preferred in Algorithm 2 with $w = 2$ whenever an inversion costs more than 6 multiplications.

Table 1: Field operation costs for point and curve operations, where M, I, and V denote field multiplication, inversion, and division, respectively. The cost H of halving is an estimate. A' denotes the cost of a point addition when one of the inputs is in λ-representation.

Calculation	Point operations	Field operations	
		affine	projective[a]
Point operation			
Addition	A	$M+V$	$8M$
Addition[b]	A'	$2M+V$	$9M$
Double	D	$M+V$	$4M$
Halve[c]	H	$2M$	
Curve operation (with width-2 NAF)			
kP via doubling	$(1/3)tA+tD$	$(4/3)t(M+V)$	$(20/3)tM+(2M+I)$
kP via Alg 2	$(1/3)tA'+tH$	$(8/3)tM+(1/3)tV$	$5tM+(2M+I)$

[a]Mixed-coordinate additions and $a \in \{0,1\}$.
[b]A field multiplication converts λ-representation to affine.
[c]Estimated.

Summary The performance advantage of halving methods is clearest in the case of point multiplication kP where P is not known in advance, and smaller field inversion to multiplication ratios generally favor halving. However, significant storage (e.g., $m/4$ field elements) for the solve routine appears to be essential for performance. It should be noted, however, that the precomputation for the solve and square root routines is per field.

References

[1] M. Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, Lecture Notes in Computer Science 1880, Springer-Verlag, 2000. 20th Annual International Cryptology Conference, Santa Barbara, California, August 2000.

[2] FIPS 186-2, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology, 2000.

[3] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited", *IEEE Transactions on Computers*, to appear.

[4] E. Knudsen, "Elliptic scalar multiplication using point halving", *Advances in Cryptology—ASIACRYPT '99*, Lecture Notes in Computer Science 1716 (1999), 135-149.

[5] D. Knuth, *The Art of Computer Programming—Seminumerical Algorithms*, Addison-Wesley, 3rd edition, 1998.

[6] R. Schroeppel, "Elliptic curves: Twice as fast!", presentation at the CRYPTO 2000 [1] Rump Session, 2000.

[7] R. Schroeppel, "Elliptic curve point ambiguity resolution apparatus and method", International Application Number PCT/US00/31014, filed 9 November 2000.

[8] J. Solinas, "Efficient arithmetic on Koblitz curves", *Designs, Codes and Cryptography* 19 (2000), 195-249.