Some Improvements on Signed Window Algorithms for Scalar Multiplications in Elliptic Curve Cryptosystems

San C. Vo
NASA Ames Research Center
Numerical Aerospace Simulation (NAS) Systems Division
svo@mail.arc.nasa.gov

Abstract

Scalar multiplication is an essential operation in elliptic curve cryptosystems because its implementation determines the speed and the memory storage requirements. This paper discusses some improvements on two popular signed window algorithms for implementing scalar multiplications of an elliptic curve point - Morain-Olivos's algorithm and Koyama-Tsuruoka's algorithm.

Keywords

Elliptic Curve Cryptosystem, Scalar Multiplication, Double-and-Add Algorithm, Addition-Subtraction Algorithm, NAF, m-ary Algorithm, Sliding Window Algorithm, Signed Window Algorithm, Morain-Olivos's Algorithm, Koyama-Tsuruoka's Algorithm.

Public key cryptography based on elliptic curves defined over Galois fields (i.e. finite fields) was proposed independently by V. Miller and N. Koblitz in 1985. Since then, elliptic curves over finite fields have been used in many applications of cryptography, such as digital signatures (e.g. Standard ANSI X9.62-1999 [1]) and key agreement and transport (e.g., working draft ANSI X9.63-200*x* [2]).

Scalar multiplication is an essential operation in an elliptic curve cryptosystem because its implementation determines the speed and the storage requirements of elliptic curve cryptosystems. Several algorithms on this operation have been described in cryptography literature. The paper discusses some improvements that were developed for two popular signed window algorithms for implementing scalar multiplications of an elliptic curve point - Morain-Olivos's and Koyama-Tsuruoka's algorithms.

The first three sections provide a review of some basic information on the scalar multiplication operation in an elliptic curve cryptosystem and the signed window algorithms. Section 4 reviews Morian-Olivos's algorithm and shows a generalization of the algorithm for more general patterns of the scalar term involved in the scalar multiplication. Section 5 provides a review of Koyama-Tsuruoka's algorithm and show how to improve its applications for a more general binary expansion of the scalar term in a scalar multiplication operation.

1. Scalar multiplication in elliptic curve cryptosystems.

Given a point $P = (x,y)$ on an elliptic curve E and an integer k, the scalar multiplication kP is defined by the recursive addition:

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ terms}}, \text{ if } k > 0, \text{ and } kP = (-k)(-P), \text{ if } k < 0.$$

When $k = 0$, then $0P = O$, the point at infinity. Since the inverse point $-P$ is obtainable at almost no cost, as is shown in the following table, from now on, k may be assumed to be positive.

$y^2 = x^3 + ax + b$ (over F_p where prime $p \neq 2, 3$)	$-P = (x,-y)$
$y^2 + xy = x^3 + ax^2 + b$ (non-supersingular elliptic curve over F_{2^m})	$-P = (x, y + x)$
$y^2 + cy = x^3 + ax + b$ (supersingular elliptic curve over F_{2^m})	$-P = (x, y + c)$

Table 1. Additive inverse of an elliptic curve point $P = (x, y)$

Scalar multiplication is the analog to raising an element to the k-th power of a number, modulo n. The research problem is to compute the elliptic curve point kP with the minimum number of addition operations. The information in Table 1 will be used when dealing with subtraction. The subtraction by a point P is simply addition with its additive inverse point $-P$.

2. Some basic algorithms for implementing scalar multiplications

This section gives a brief review of some basic algorithms available in cryptography literature and implementations. More detailed information can be found in the referenced papers.

2.1. The most basic method is the double-and-add method that uses the binary expansion of the integer k. Let $k = (k_{r-1},...,k_o)$ in base 2, where $r = \lfloor \log_2 k \rfloor + 1$. Let $P_{r-2} = P$. Compute $P_{i-1} = 2P_i + k_i P$, for all i, $r-2 \geq i \geq 0$. This produces $kP = P_{-1} = 2P_0 + k_0 P$. This method requires $(r-1)$ doublings and, probabilistically, about $(r-1)/2$ additions. Observe that the number of arithmetic operations can be reduced when the number of bits 0 is increased. This is the basic idea for methods that try to improve on the implementation of scalar multiplications.

2.2. An improved method for implementing scalar multiplication is the addition-subtraction method. For elliptic curve implementations, the methods that include subtractions are more attractive than the corresponding methods that include divisions in calculating powers of an element in finite fields. The reason is that division or inversion in finite fields is a more costly operation than multiplication, while subtraction is just as costly as addition in elliptic curve operations.

An addition-subtraction method is recommended in the Standard ANSI X9.62-1999 [1] and the working draft for ANSI X9.63 [2] that uses the binary expansion of k and $3k$. Let $l = 3k = (l_{r-1},...,l_o)$ and $k = (k_{r-1},...,k_o)$ such that the leftmost bit l_{r-1} must be 1. Let $P_{r-1} = P$. Compute $P_{i-1} = 2P_i + (l_{i-1} - k_{i-1})P$, for all i, $r-1 \geq i \geq 2$, resulting in $kP = P_1 = 2P_2 + (l_1 - k_1)P$. This method requires $(r-2)$ doublings and, probabilistically, about $(r-2)/2$ additions/subtractions.

2.3. Another improved version is called the m-ary (or 2^d-ary) method. In this method, the m-ary expansion is used instead of the binary form, where $m = 2^d$, and $d > 1$. The

binary expansion of the scalar $k = (k_{s-1},...,k_o)$ is padded with an extra number of bits 0 to the left side of the bit string, if necessary, to make $s = dr$ for some integer r. This produces the m-ary expansion of k of the form $k = (K_{r-1},...,K_o)$, where each K_i is a d-bit string, and $K_{r-1} \neq (0\cdots0)$. Pre-compute all points $2P$, $3P$,..., $(2^d - 1)P$. These points will include all possible points of the form K_iP. The algorithm is similar to the basic binary algorithm. This method requires $(2^d - 2)$ pre-computations (and hence memory storage), $(r-1)d = s - d$ doublings and, probabilistically, about $(r-1)(1-2^{-d})$ additions. Observe that the larger d is, the more pre-computations are needed. With a little calculus, the optimal value of d can be found that will minimize the total number of additions.

2.4. Another algorithm is the addition-subtraction method using non-adjacent form (NAF). The canonical non-adjacent form of the scalar k employs a signed binary expansion (using 0 and ± 1) that has the property that no two consecutive coefficients are nonzero. The NAF of an integer is unique and has the fewest nonzero digits of any signed binary expansions. There are a few ways to construct the NAF. The addition-subtraction method requires $(r-1)$ doublings and, probabilistically, about $(r-1)/3$ additions.

It is possible that the addition-subtraction method can be combined with the 2^d-ary method. In particular, the addition-subtraction method using the 2^d-ary NAF form is a binary form with the property that there is at most one non-zero term in d consecutive coefficients. This form always uniquely exists and is easy to compute. The computing method is similar to that for the NAF form, except that the corresponding quotient corresponding to the non-zero remainder must be divisible by 2^{d-1}. The pre-computation process must store the values of all points: $\pm P$, $\pm 3P$,..., $\pm(2^{d-1} - 1)P$.

3. Signed window algorithms

More advanced methods for implementing scalar multiplications are referred to as window methods. The basic one is called the Sliding window method. (Refer to [6]). This method's purpose is to separate zero words so that an addition in the m-ary method discussed above can be skipped. (A zero word is defined as a bit string of only 0's.)

Instead of decomposing $k = (k_{s-1},...,k_o)$ into d-bit length words of fixed d, decompose k into zero and nonzero words, or windows W_i of varying lengths l_i. Let d be the maximum length of all nonzero windows. Then pre-compute only the points of odd scalar multiples: $3P$, $5P$,..., $(2^d - 1)P$. Let $k = (W_{r-1},...,W_o)$, where W_{r-1} is a non-zero window (or window number). There are two strategies to partition a binary expansion into windows: constant length nonzero windows and variable length nonzero windows.

- Constant length nonzero windows: This strategy tries to produce zero windows of arbitrary length and nonzero windows of a fixed length d. A nonzero window will start when a bit 1 is encountered as the bits are scanned from the rightmost bit to the leftmost bit.
- Variable length nonzero windows: This strategy tries to produce nonzero windows whose right-end and left-end bits are both 1. Two parameters are to be chosen optimally in order to decrease the average number of nonzero windows:

the maximum length d of nonzero windows and the maximum number r of adjacent 0's allowed inside any nonzero window.

Signed binary window algorithms are more advanced. Their strategy is to transform a binary expansion into a signed binary expansion. The algorithms using "signed" expansion are much more efficient in implementing elliptic curve operations than in implementing the modular exponentiation operations, since subtraction is just as costly as addition. The purpose of these algorithms is to skip a bit string of all 1's to reduce the number of additions. Two popular algorithms mentioned in cryptography literature are reviewed in the following sections - Morain-Olivos's and Koyama-Tsuruoka's algorithms. Improvements are developed for these two algorithms.

4. Morain-Olivos's algorithm and its improvement

4.1. Basic technique (Refer to [8])

This algorithm reduces the weight of the signed binary expansion, i.e., the number of non-zero digits in the expansion. The idea is that a block of n bits 1 can be replaced by a bit string that is a block consisting of a bit 1 followed by n bits 0 and then minus 1. That is, $\underbrace{11\cdots1}_{n \text{ bits}} = 1\underbrace{00\cdots0}_{n \text{ bits}} - 1$. This observation is extracted from an arithmetic equality, that can be easily verified: $2^{n+1} - 1 = \left(2^n + 2^{n-1} + \cdots + 2^1 + 2^0\right)$.

As a result, $(n-1)$ doublings and $(n-1)$ additions (i.e., $2(n-1)$ total additions) can be replaced by n doublings and 1 subtraction (i.e., $(n+1)$ total additions). In other words, this method tries to construct two positive integers k_+ and k_- such that $k_+ - k_- = k$. The total computation for $(k_+ - k_-)P$ is less than that for kP. Note that there are not two separate computations of k_+P and k_-P, but actually the computations merge together: k_-P only shows up in a few subtractions corresponding to the positions of its bits 1 in the scalar k.

Example: $183_{10} = \underline{1}\,\underline{011}\,\underline{0111}_2 = \underline{1}\,\underline{100}\,\underline{1000} - \underline{0}\,\underline{001}\,\underline{0001} = k_+ - k_- = \underline{1}\,\underline{10\bar{1}}\,\underline{100\bar{1}}$. Then by a binary method, $183P = 2^3(2[2^2(2P+P)-P]+P)-P$, where the two subtractions correspond exactly to the two bits 1 in the expansion of $k_- = 00010001$, or two bits $\bar{1}$. Using a sliding window method, the following can be written:

$183_{10} = \underline{10110111}_2 = \underline{11001000} - \underline{00010001} = \underline{110\bar{1}100\bar{1}}$ and $183P = 2^4(12P - P) + 8P - P$.

The same idea can deal with a special string that has isolated 0's. Observe that: $k = \underbrace{1\cdots1}_{n \text{ bits}}0\underbrace{1\cdots1}_{m \text{ bits}} = 1\underbrace{0\cdots0}_{n \text{ bits}}\bar{1}\underbrace{0\cdots0}_{(m-1) \text{ bits}}\bar{1} = 1\underbrace{0\cdots\cdots0}_{n+m+1 \text{ bits}} - 0\underbrace{0\cdots0}_{n \text{ bits}}1\underbrace{0\cdots\cdots0}_{(m-1) \text{ bits}}1$, where $\bar{1}$ denotes -1, and m is assumed to be greater than or equal to 2. This observation is extracted from the same equality described above and applied twice. Then the following formula can be obtained:

$$kP = 2^m(2^{n+1}P - P) - P.$$

That is, an isolated 0 inside a block of bits 1 will contribute only two subtractions/additions and one extra doubling, instead of $(n+m-1)$ additions.

Refer to [8] for detailed estimations of the implementation cost. In summary, the algorithm reduces the number of operations (additions and doublings) by about 3% for

100-digit numbers and by 2.7% for 300-digit numbers. Refer to the work of [4] for similar approaches.

4.2. Improvement: A generalization of Morain-Olivos's algorithm

In this section, I will generalize the Morain-Olivos's algorithm to show that it can be applied in additional cases of scalar multiplication operations. The above approach applies only to the case where the scalar $k = \underbrace{1\cdots1}_{n\text{ bits}}0\underbrace{1\cdots1}_{m\text{ bits}}$. I will show that a more general pattern of k is also applicable.

Indeed, for k being a string of $(b-1)$ isolated bits 0 sandwiched among b blocks of bits 1, where b is larger than 2, the following can be written:

$$k = \underbrace{1\cdots\cdots1}_{N_1\text{ bits}}0\cdots\underbrace{1\cdots\cdots1}_{N_{b-1}\text{ bits}}0\underbrace{1\cdots\cdots1}_{N_b\text{ bits}} = 1\underbrace{0\cdots\cdots\cdots\cdots0}_{N_1+\cdots+N_b+(b-1)\text{ bits}} - 0\underbrace{0\cdots0}_{N_1\text{ bits}}1\cdots\underbrace{0\cdots\cdots0}_{N_{b-1}\text{ bits}}1\underbrace{0\cdots\cdots0}_{(N_b-1)\text{ bits}}1,$$

assuming $N_b \geq 2$. This observation is also derived from the above equality: $2^{n+1}-1 = \left(2^n + 2^{n-1}+\cdots+2^1+2^0\right)$. Then the following formula can be obtained:

$$kP = 2^{N_b}(2^{N_{b-1}+1}[\cdots(2^{N_1+1}P - P)\cdots] - P) - P.$$

This formula generalized dramatically the application of Morain-Olivos's algorithm for the case where b = 2 only to the case where b is any positive number that is greater than 2.

By applying this algorithm, $(N_1+\cdots+N_b-1)$ additions were replaced by only b subtractions/additions and one extra doubling. The savings in the number of arithmetic operations are significant when the sum $(N_1+\cdots+N_b)$ is much larger than b, which should be obtainable for those cases of k.

Example: $183_{10} = \underline{1}\,0\,\underline{11}\,0\,\underline{111}_2 = 100000000 - 0\underline{01}\,\underline{001}\,\underline{001} = 10\bar{1}00\bar{1}00\bar{1}$. Then

$$183P = 2^3[2^3(2^2P - P) - P] - P.$$

$1467_{10} = \underline{1}0\underline{11}0\underline{111}0\underline{11}_2 = 100000000000 - 0\underline{01}00\underline{1}000\underline{1}0\underline{1} = 10\bar{1}00\bar{1}000\bar{1}0\bar{1}$. Then

$$1467P = 2^2(2^4[2^3(2^2P - P) - P] - P) - P.$$

This improvement expands the domain of applicable patterns in the binary expansion of the scalar k in the Morain-Olivos's algorithm. This generalized algorithm can be improved dramatically if the direct multiplication formulae (refer to [3]) can be employed to calculate the points 2^dP, where d is up to the maximum number N_i needed in the above formula for kP.

5. Koyama-Tsuruoka's algorithm and its improvement

5.1. Basic technique (Refer to [7])

The Koyama-Tsuruoka's algorithm attempts to increase the average length of zeros and decrease the weight in the signed binary expansion using $\{\bar{1},0,1\}$, where $\bar{1}$ denotes -1. Denote the number of bits 0 (and, respectively, bits 1) in a bit string B by $\#_0(B)$ (and, respectively, $\#_1(B)$).

A binary string of a non-zero window $B = (1,b_n,...,b_1,1)$ in k will be transformed to a signed binary string of the form $T = (1,0,t_n,...,t_1,\bar{1})$, where $t_i = b_i - 1$, for all $1 \leq i \leq n$.

This transformation is effective (i.e. actually decreases the weight of the bit string) only when the difference between the numbers of bits 1 and bits 0 in B is: $Diff(B) = \#_1(B) - \#_0(B) > 2$. However, keep in mind that the transformation also costs one extra doubling because of the extra bit. This transformation is extracted from an arithmetic equality that is also easily verified:

$$2^{n+2} - 2^{n+1} = 2^{n+1} = \left(2^n + 2^{n-1} + \cdots + 2^1 + 1 + 1\right).$$

Using the relationship $1 = b_i - t_i$, for all $1 \le i \le n$, then

$$2^{n+2} + \left(t_n 2^n + t_{n-1} 2^{n-1} + \cdots + t_1 2^1 + 1\right) = 2^{n+1} + \left(b_n 2^n + b_{n-1} 2^{n-1} + \cdots + b_1 2^1 + 1\right).$$

Using this method, one needs to pre-compute only $\pm 3P, \pm 5P, ..., \pm(2^d - 3)P$, since the algorithm never allows terms of the form $\pm(2^d - 1)P$ to appear.

Example: $183_{10} = \underline{1}\,0\,\underline{11}\,0\,\underline{111}_2 = \underline{1}\,0\,\underline{11}\,100\bar{1}$. Then $183P = 16(11P) + 7P$.

$743_{10} = \underline{10111}\,\underline{00111}_2 = \underline{10\bar{1}00\bar{1}}\,\underline{0100\bar{1}}$. Then by a sliding window method with $d = 5$,

$$743P = 2^5(2^5 P - 9P) + 7P.$$

A similar idea to Koyama-Tsuruoka's algorithm was also discussed in [5]. Refer to [7] for a detailed performance evaluation and comparison.

5.2. An Improvement of Koyama-Tsuruoka's algorithm

The Koyama-Tsuruoka's algorithm is applied for the non-zero window $(1, b_n, ..., b_1, 1)$ only. This limits its application. In this section, I will show how to significantly improve this algorithm for any binary expansion.

Instead of working only on a non-zero window, consider an arbitrary bit string $k = (b_n, ..., b_1, b_0)$, where, without loss of generality, $b_n = 1$ can be assumed. For all $0 \le i \le n$, let $t_i = b_i - 1$, then insert the relation $1 = t_i + b_i$ into the arithmetic identity: $2^{n+1} = \left(2^n + 2^{n-1} + \cdots + 2^1 + 2^0\right) + 1$, to obtain the following identity:

$$2^{n+1} + \left(t_n 2^n + t_{n-1} 2^{n-1} + \cdots + t_1 2^1\right) + t_0 2^0 - 1 = \left(b_n 2^n + b_{n-1} 2^{n-1} + \cdots + b_1 2^1 + b_0 2^0\right).$$

When $b_0 = 1$, then $t_0 = 0$. This approach will transform k into the string of digits $T = (1, t_n, ..., t_1, \bar{1})$, where the last digit $\bar{1} = -1$. This is Koyama-Tsuruoka's algorithm for a non-zero window $B = (b_n, ..., b_1, 1)$.

When $b_0 = 0$, then $t_0 = -1$. This approach will transform $k = (b_n, ..., b_1, 0)$ into the string of digits $T = (1, t_n, ..., t_1, \bar{2})$, where the last digit $\bar{2} = -2$. This last digit does not affect the scalar multiplication (kP) at all. In the very last step of a given scalar multiplication algorithm, subtract a double of a point $2P$ instead of the point P itself. The point $2P$ is available for free since it is always computed during the process. If other m-ary methods or window methods are used, the digit $\bar{2}$ is obviously not a concern anyway. Only some minor changes are needed in the pre-computations.

Example: $742_{10} = 1011100110_2 = 10\bar{1}000\bar{1}\,\bar{1}00\bar{2}_2$. Then by a sliding window method,

$743P = 2^5(2^5 P - 8P) - 26P$.

The number of non-zero digits in T is

$$\#_{non-0}(T) = 2 + \sum_{i=1}^{n} |t_i| = 2 + \sum_{i=1}^{n} |b_i - 1| = 2 + \sum_{1 \le i \le n, b_i = 0} 1 = 2 + \#_0(k'), \text{ where } k' = (b_n, \ldots, b_1).$$

Hence, this transformation is effective if the condition $2+\#_0(k') < \#_1(k)$ is satisfied. The transformation costs one extra doubling because of the extra bit. Since $\#_0(k') = \#_0(k) - 1$, the condition can be rewritten as: $Diff(k) = \#_1(k) - \#_0(k) > 1$.

Therefore, Koyama-Tsuroka's algorithm can now be applied to any binary bit strings that satisfy the previous condition on *Diff*(*k*). The improved algorithm dramatically expands the application of the original algorithm by removing the limit of applying only on non-zero window pattern.

6. Conclusions

The intention is to modify or to develop methods using some digits other than just bits 0 and 1 in the standard binary expansion of the scalar k of the scalar multiplication. The benefit of using the 2 digit in the expansion is because of the availability of the point $2P$ in the (pre-)computation processes. Using "negative" digits (e.g., $\bar{1} = -1$ or $\bar{2} = -2$) has great benefits since the cost of subtraction is the same as that of addition in elliptic curve implementations. This is a considerable improvement to the approaches for developing new algorithms to improve the efficiency of implementations of scalar multiplication operations in elliptic curve cryptosystems.

This paper presented two improvements on two popular signed window algorithms for scalar multiplications implemented in elliptic curve cryptosystems. The first improvement on Morain-Olivos's algorithm helped to expand its domain of applicable patterns in the binary expansion of the scalar k. The improvement on Koyama-Tsuruoka's algorithm expands the application domain from non-zero windows to any binary expansion of k satisfying the condition on the difference on the numbers of bits 0 and 1. These two improvements facilitate the efficiency of implementation of scalar multiplication in elliptic curve cryptosystems.

References

[1] ANSI X9.62-1999, Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA), American Bankers Association, 1999.

[2] ANSI X9.63-200*x*, Public key cryptography for the financial services industry: Elliptic curve key agreement and transport protocol, American Bankers Association, 2000.

[3] J. Guajargo & C. Paar, Efficient algorithms for elliptic curve cryptosystems, Advances in Cryptology - Crypto '97, Lecture Notes in Computer Science 1294, 342-356, Springer-Verlag, 1997.

[4] J. Jedwab & C. Mitchell, Minimum weight modified signed-digit representations and fast exponentiation, Electronics Letters, Vol. **25**, No. 17 (1989), 1171-1172.

[5] C. Koç, High-radix and bit recoding techniques for modular exponentiation, International Journal of Computer Mathematics, Vol. 40, 1991, 139-156.

[6] C. Koç, Analysis of sliding window techniques for exponentiation, Computers Math. Applications, Vol. 30, No. 10, 1995, 17-24.

[7] K. Koyama & Y. Tsuruoka, Speeding up elliptic cryptosystems by using a signed binary window method, Advances in Cryptology - Crypto '92, Lecture Notes in Computer Science 740, 345-357, Springer-Verlag, 1993.

[8] F. Morain & J. Olivos, Speeding up the computations on an elliptic curve using addition-subtraction chains, RAIRO-Informatique Théorique et Applications - Theoretical Informatics and Applications, Vol. 24, No. 6, 1990, 531-544.