Math-Net.Ru

Общероссийский математический портал

S. V. Grebnev, Оптимизация объема памяти при мультискалярном умножении,
Матем. вопр. криптогр., 2016, том 7, выпуск 2, 53–60

DOI: <https://doi.org/10.4213/mvk183>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и
согласны с пользовательским соглашением

<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 54.174.149.18

26 апреля 2023 г., 17:52:01

Optimizing memory cost of multi-scalar multiplication

S. V. Grebnev

National Research University Higher School of Economics, Moscow

Получено 01.III.2015

Abstract. We present a modification of the non-adjacent form representation for multi-scalar multiplication which allows efficient performance-memory tradeoffs. Simultaneously we fix a feature of the original algorithm which prevented it from using all possible bases.

Keywords: multi-scalar multiplication, non-adjacent form, digital signature, performance evaluation

Оптимизация объема памяти при мультискалярном умножении

С. В. Гребнев

Национальный исследовательский университет

Высшая школа экономики, Москва

Аннотация. Предлагается модификация разреженного представления для мультискалярного умножения, обеспечивающая эффективную балансировку эффективности и памяти. Кроме того, выявлена особенность исходного алгоритма, препятствующая использованию всех возможных оснований.

Ключевые слова: мультискалярное умножение, разреженное представление, цифровая подпись, оценка эффективности

1. Introduction

Multi-scalar multiplication, i.e. computation of the form $k_1P + k_2Q$, where P and Q are elements of a finite group and k_1, k_2 are integers, is an essential part of many cryptographic primitives. Consider, for example, Russian national digital signature standard scheme GOST R 34.10-2012. The scheme is constructed over a subgroup of points $\langle P \rangle$ of an elliptic curve E over a prime field, where $\#\langle P \rangle \approx 2^{256}$ or $\#\langle P \rangle \approx 2^{512}$. The signature verification process is defined by the equation

$$x_{(s h(M) \bmod q)P - (r h(M) \bmod q)Q} \pmod{q} = r,$$

where (r, s) is a signature, $h(M)$ is a hashed message, and $Q = dP$ is the public key; the signature is valid if and only if the equality holds. Thus, improving efficiency of multi-scalar multiplication is crucial for implementations of the signature scheme.

In the previous paper [2] we have considered several possibilities of improving performance of GOST R 34.10-2012, including state-of-the-art choice of an elliptic curve representation and optimisation of scalar and multi-scalar multiplication. Recent paper deals with multi-scalar multiplication exclusively. We recall an algorithm suggested by M. Kalinin which was sketched in [2], fix its minor features and present a novel variant of joint multibase form which allows for more flexible memory management.

2. Multi-scalar multiplication

The computation of $k_1P + k_2Q$ may be performed as two successive scalar multiplications k_1P , k_2Q and an addition. However, it is possible to use so-called Shamir's trick, that is, jointly scan binary representations of k_1, k_2 , performing at each step a doubling and additions of P , Q or $P + Q$, if the corresponding bits are set, and save on excessive multiplications. We can do even better, using specially optimized sparse representations to save on additions.

A common scheme for multi-scalar multiplication may be described as follows:

- 1) item pre-compute a set of points iP, jQ , where i, j belong to a method-specific set of integers,
- 2) construct a special joint representation of the pair (k_1, k_2) ,
- 3) set the result to infinity, then "scan" the representation of exponents, adding a precomputed point depending on the corresponding digit and multiplying by the bases associated with the digit.

The most commonly used method is the joint sparse form, introduced in [9].

The development of elliptic curve cryptosystems gave rise to extensive study of their equivalent representations, such as Edwards curves, Hesse curves etc., see [3, 4]. Many of these allow to do basic operations faster, e. g., doublings and triplings. In order to use their advantages in scalar and multi-scalar multiplications, a number of special representations was designed, to mention joint double-base number system and joint double-base chains [1].

Window-based multibase representations (proposed in [6, 8] for scalar multiplications) were later adapted for multi-scalar multiplications in [5]. We proceed with their description.

Let $A = \{a_1, a_2, \dots, a_J\}$ be a fixed finite set of coprime positive integers. Fix a parameter w , traditionally called a *window*, and define the *multi-base non-adjacent form with window w* (wmbNAF) of an integer k , denoted by $(a_1, a_2, \dots, a_J)NAF_w(k) = \{d_1^{(a_1)} d_2^{(a_2)} \dots d_m^{(a_m)}\}$, as a sequence of digits d_i associated with corresponding bases $a_i \in A$ such that:

- 1) $k = (\dots((d_1 \cdot (a_2) + d_2) \cdot (a_3)) + \dots + d_{m-1}) \cdot (a_m) + d_m$,
- 2) $d_i \in \{0, \pm 1, \pm 2, \dots, \pm \lfloor \frac{a_i^w - 1}{2} \rfloor\} \setminus \{\pm a_1, \pm 2a_1, \dots, \pm \lfloor \frac{a_1^{w-1} - 1}{2} \rfloor a_1\}$, $d_1 > 0$,
- 3) of any w consecutive digits, at most one is non-zero,
- 4) any positive integer k has a unique representation of the form $(a_1, a_2, \dots, a_J)NAF_w(d)$.

Note that we keep the original notations from [6, 8] hereafter, so the term $d_i = d^{(a_j)}$ should be read as “set the i -th digit to the value d and set the associated base to a_j ”. The reader should not be confused by the notation (a_i) , which denotes the base actually written at the i -th position of the wmbNAF, *not* its index in A .

A *joint multibase non-adjacent form with window w* (JNAF) of a pair of integers k_1, k_2 is actually a pair of wmbNAFs of length l , maybe left-padded with zeroes for a smaller integer, calculated for k_1, k_2 simultaneously in such a manner that the bases (a_i) associated with each position coincide:

$$(a_1, a_2, \dots, a_J)NAF_w(k_1, k_2) = \left(\begin{pmatrix} d_1^{(a_1)}, d_2^{(a_2)}, \dots, d_l^{(a_l)} \\ e_1^{(a_1)}, e_2^{(a_2)}, \dots, e_l^{(a_l)} \end{pmatrix} \right),$$

and

$$k_1 P + k_2 Q = (a_{l-1})(\dots(a_3)((a_2)(d_1 P + e_1 Q) + d_2 P + e_2 Q) + \dots + d_{l-1} P + e_{l-1} Q) + d_l P + e_l Q;$$

the latter formula gives a natural algorithm for multi-scalar multiplication.

The paper [5] presents algorithms to compute JNAF in two slightly different ways, denoted $(a_1, a_2, \dots, a_J) \overrightarrow{\text{NAF}}_w(k_1, k_2)$ (“left-to-right”) and $(a_1, a_2, \dots, \dots, a_J) \overleftarrow{\text{NAF}}_w(k_1, k_2)$ (“right-to-left”) depending on the order in which the bases are treated. Since the paper [5] is not widely available, we provide a sketch of the algorithm. We denote by ka a residue of k modulo a such that $ka \in [-a/2, a/2)$.

Algorithm 1.

Input: $k_1, k_2 \in \mathbb{N}$, $J \geq 1$, $A = \{a_1, a_2, \dots, a_J\}$ — A SET OF ASCENDING DIFFERENT PRIME BASES, $j = 1, 2, \dots, J$, A WINDOW $w \in \mathbb{N}$, $w \geq 2$.

Output: $(a_1, a_2, \dots, a_J) \overrightarrow{\text{NAF}}_w(k_1, k_2) = \left((d_1^{(a_1)}, d_2^{(a_2)}, \dots, d_l^{(a_l)}) \right)$
 $(e_1^{(a_1)}, e_2^{(a_2)}, \dots, e_l^{(a_l)})$

1. $i = 1$;
2. WHILE $k_1 > 0$ OR $k_2 > 0$
 - 2.1. FOR $j = 1$ TO J DO % (“Left-to-right” case)
OR
FOR $j = J$ DOWNTO 1 DO % (“Right-to-left” case)
 - 2.1.1. IF $((k_1 \bmod a_j \equiv 0) \text{ AND } (k_2 \bmod a_j \equiv 0))$ THEN
 - 2.1.1.1. $d_i = 0$, $k_1 = k_1/a_j$, $d_i = d_i^{(a_j)}$;
 - 2.1.1.2. $e_i = 0$, $k_2 = k_2/a_j$, $e_i = e_i^{(a_j)}$;
 - 2.1.1.3. BREAK;
 - 2.1.2. ELSE IF $((k_1 \bmod a_j \equiv 0) \text{ AND } (k_2 \bmod a_j \not\equiv 0))$ THEN
 - 2.1.2.1. $e_i = k_2 a_1^w$,
 - 2.1.2.2. $k_2 = k_2 - e_i$;
 - 2.1.2.3. IF $(k_2 \bmod a_j \equiv 0)$ THEN
 - 2.1.2.3.1. $d_i = 0$, $d_i = d_i^{(a_j)}$;
 - 2.1.2.3.2. $e_i = e_i^{(a_j)}$;
 - 2.1.2.3.3. $k_1 = k_1/a_j$, $k_2 = k_2/a_j$;
 - 2.1.2.3.4. BREAK;
 - 2.1.2.4. ELSE $k_2 = k_2 + e_i$;
 - 2.1.3. ELSE IF $((k_1 \bmod a_j \not\equiv 0) \text{ AND } (k_2 \bmod a_j \equiv 0))$ THEN
 - 2.1.3.1. $d_i = k_1 a_1^w$,
 - 2.1.3.2. $k_1 = k_1 - d_i$;
 - 2.1.3.3. IF $(k_1 \bmod a_j \equiv 0)$ THEN
 - 2.1.3.3.1. $e_i = 0$, $e_i = e_i^{(a_j)}$;
 - 2.1.3.3.2. $d_i = d_i^{(a_j)}$;
 - 2.1.3.3.3. $k_1 = k_1/a_j$, $k_2 = k_2/a_j$;
 - 2.1.3.3.4. BREAK;
 - 2.1.3.4. ELSE $k_1 = k_1 + d_i$;

```

2.2. IF  $j > J$  THEN
    2.2.1.  $d_i = k_1 a_1^w$ ;
    2.2.2.  $k_1 = k_1 - d_i$ ;
    2.2.3.  $e_i = k_2 a_1^w$ ;
    2.2.4.  $k_2 = k_2 - e_i$ ;
    2.2.5. FOR  $k = 1$  TO  $J$  DO % ("Left-to-right" case)
        OR
        FOR  $k = J$  DOWNTO 1 DO % ("Right-to-left" case)
    2.2.5.1 IF  $((k_1 \bmod a_k \equiv 0) \text{ AND } (k_2 \bmod a_k \equiv 0))$  THEN
        2.2.5.1.1.  $d_i = d_i^{(a_k)}$ ;
        2.2.5.1.2.  $e_i = e_i^{(a_k)}$ ;
        2.2.5.1.3.  $k_1 = k_1/a_k, k_2 = k_2/a_k$ ;
        2.2.5.1.4. BREAK;
2.3.  $i = i + 1$ ;
3. RETURN  $(a_1, \dots, a_J) \overrightarrow{\text{NAF}}_w(k_1, k_2) = \left( \begin{pmatrix} d_1^{(a_1)}, d_2^{(a_2)}, \dots, d_l^{(a_l)} \\ e_1^{(a_1)}, e_2^{(a_2)}, \dots, e_l^{(a_l)} \end{pmatrix} \right)$ .

```

Remark. Note that the description of the algorithm given in [2] is inaccurate, since in the case of “left-to-right” computation the cycle (step 2.1) was continuously performed for $j=1$, thus, all bases greater than a_1 were ignored, and the output was always of the form (2) $\overrightarrow{\text{NAF}}_w(k_1, k_2)$. In the version presented here this inaccuracy is corrected.

Given a JNAF, we consider two algorithms for computing $k_1P + k_2Q$, using either precomputed points iP, jQ or $iP \pm jQ$. As is shown in [5], the digits of $(a_1, a_2, \dots, a_J) \overrightarrow{\text{NAF}}_w(k_1, k_2)$ are such that

$$d_i, e_i \in \left\{ 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{a_1^w - 1}{2} a_1 \right\rfloor \right\} \setminus \left\{ \pm a_1, \pm 2a_1, \dots, \left\lfloor \frac{a_1^{w-1} - 1}{2} \right\rfloor \right\},$$

thus $a_1^w/2$ or $a_1^w/4$ precomputed points are required, depending on their form, while the digits of $(a_1, a_2, \dots, a_J) \overrightarrow{\text{NAF}}_w(k_1, k_2)$ are such that

$$d_i, e_i \in \left\{ 0, \pm 1, \pm 2, \dots, \pm \left\lfloor \frac{a_1^w}{2} \right\rfloor \right\} \setminus \{\pm 6k, k \in \mathbb{N}\},$$

so $\lfloor 5a_1^w/12 \rfloor$ or $\lfloor 25a_1^w/144 \rfloor^2$ precomputed points are required. Obviously, the algorithms with $iP \pm jQ$ precomputations require too much storage to be efficient in practice. Even for the tables with iP, jQ it may be desirable to reduce storage requirements while keeping performance characteristics. We proceed with the description of proposed solution.

3. Joint fractional multibase non-adjacent form

In 2008, Longa and Gebotys presented *fractional multibase non-adjacent form* which, given a window size w and a fixed set $\mathcal{D} = \{\pm 1, \pm 3, \dots, \pm (2t+1)\}$ of digits, produced a flexible encoding of an integer in a NAF_w -form, where all the digits are in $\mathcal{D}$. We describe an adjustment of this representation for the case of joint multibase non-adjacent form.

Suppose we have a set of J bases $\{2, a_2, \dots, a_J\}$ (that is $a_1 = 2$). Next, for $d \in \{\pm 1, \pm 3, \dots, \pm m\}$, where $m \geq 3$, m is an odd integer, we precompute $\{d_P\}$, $\{d_Q\}$. Fix a window w and let $m = 2^{w-2} + s$ for an odd $s \geq 1$.

Following [7], we write the recoding rule for our representation.

1. IF $k \equiv 0 \pmod{a_1}$ OR $k \equiv 0 \pmod{a_2}$ OR $\dots$ OR $k \equiv 0 \pmod{a_J}$
THEN $k_i = 0$;
2. ELSE IF $0 < r \leq m$ THEN $k_i = r$;
3. ELSE IF $m \leq r < (3m - 4s)$ THEN $k_i = r - 2^{w-1}$;
4. ELSE IF $(3m - 4s) \leq r < 2^w$ THEN $k_i = r - 2^w$,

where $r = k \pmod{2^w}$. Now we give a complete description of the algorithm.

Algorithm 2.

Input: $k_1, k_2 \in \mathbb{N}$, $\mathcal{A} = \{a_1, a_2, \dots, a_J\}$, $J \geq 1$, IS A SET OF ASCENDING PRIME BASES WITH $a_1 = 2$, A WINDOW $w \in \mathbb{N}$, $w \geq 2$, A SET OF DIGITS $\mathcal{D} = \{\pm 1, \pm 3, \dots, \pm m\}$ SUCH THAT $m = 2^{w-2} + s$, $2^{w-2} < m < 2^{w-1}$, m, s ARE ODD POSITIVE INTEGERS.

Output: JOINT FRACTIONAL NON-ADJACENT FORM

$$(a_1, a_2, \dots, a_J) \overrightarrow{\text{NAF}}_{w,t}(k_1, k_2) = \begin{pmatrix} (d_1^{(a_1)}, d_2^{(a_2)}, \dots, d_l^{(a_l)}) \\ (e_1^{(a_1)}, e_2^{(a_2)}, \dots, e_l^{(a_l)}) \end{pmatrix},$$

WHERE $d_i, e_i \in \{\pm 1, \pm 3, \dots, \pm m\}$.

1. $i = 0$;
2. WHILE $(k_1 > 0)$ OR $(k_2 > 0)$ DO
 - 2.1. FOR $j = 1$ TO J DO
 - 2.1.1. IF $((k_1 \equiv 0 \pmod{a_j}) \text{ AND } (k_2 \equiv 0 \pmod{a_j}))$ THEN
 - 2.1.1.1. $d_i = 0^{(a_j)}$, $d_j = 0^{(a_j)}$;
 - 2.1.1.2. $k_1 = k_1/a_j$, $k_2 = k_2/a_j$;
 - 2.1.1.3. BREAK;
 - 2.1.2. ELSE IF $((k_1 \equiv 0 \pmod{a_j}) \text{ AND } (k_2 \equiv 0 \pmod{a_j}))$ THEN
 - 2.1.2.1. $r \equiv k_2 \pmod{2^w}$;
 - 2.1.2.2. IF $0 < r \leq m$ THEN $e_i = r$;
 - 2.1.2.3. ELSE IF $m < r \leq (3m - 4s)$ THEN $e_i = r - 2^{w-1}$;
 - 2.1.2.4. ELSE IF $(3m - 4s) < r < 2^w$ THEN $e_i = r - 2^w$;
 - 2.1.2.5. $k_2 = k_2 - e_i$;

```

2.1.2.6. IF  $k_2 \equiv 0 \pmod{a_j}$  THEN
    2.1.2.6.1.  $d_i = 0^{(a_j)}$ ,  $e_i = e_i^{(a_j)}$ ;
    2.1.2.6.2.  $k_1 = k_1/a_j$ ,  $k_2 = k_2/a_j$ ;
    2.1.2.6.3. BREAK;
2.1.2.7. ELSE  $k_2 = k_2 + e_i$ ;
2.1.3. ELSE IF  $((k_1 \equiv 0 \pmod{a_j}) \text{ AND } (k_2 \equiv 0 \pmod{a_j}))$  THEN
    2.1.3.1.  $r = k_1 \pmod{2^w}$ ;
    2.1.3.2. IF  $0 < r \leq m$  THEN  $d_i = r$ ;
    2.1.3.3. ELSE IF  $m < r \leq (3m-4s)$  THEN  $d_i = r - 2^{w-1}$ ;
    2.1.3.4. ELSE IF  $(3m-4s) < r < 2^2$  THEN  $d_i = r - 2^w$ ;
    2.1.3.5.  $k_1 = k_1 - d_i$ ;
    2.1.3.6. IF  $k_1 \equiv 0 \pmod{a_j}$  THEN
        2.1.3.6.1.  $d_i = d_i^{(a_j)}$ ,  $e_i = 0^{(a_j)}$ ;
        2.1.3.6.2.  $k_1 = k_1/a_j$ ,  $k_1 = k_1/a_j$ ;
        2.1.3.6.3. BREAK;
    2.1.3.7. ELSE  $k_1 = k_1 + d_i$ ;
2.2. IF  $j > J$  THEN
    2.2.1.  $r \equiv k_2 \pmod{2^w}$ ;
    2.2.2. IF  $0 < r \leq m$  THEN  $e_i = r$ ;
    2.2.3. ELSE IF  $m < r \leq (3m-4s)$  THEN  $e_i = r - 2^{w-1}$ ;
    2.2.4. ELSE IF  $(3m-4s) < r < 2^2$  THEN  $e_i = r - 2^w$ ;
    2.2.5.  $k_2 = k_2 - e_i$ ;
    2.2.6.  $r \equiv k_1 \pmod{2^w}$ ;
    2.2.7. IF  $0 < r \leq m$  THEN  $d_i = r$ ;
    2.2.8. ELSE IF  $m < r \leq (3m-4s)$  THEN  $d_i = r - 2^{w-1}$ ;
    2.2.9. ELSE IF  $(3m-4s) < r < 2^2$  THEN  $d_i = r - 2^w$ ;
    2.2.10.  $k_1 = k_1 - d_i$ ;
    2.2.11. FOR  $l = 1$  TO  $J$  DO
        2.2.11.1. IF  $((k_1 \equiv 0 \pmod{a_l}) \text{ AND } (k_2 \equiv 0 \pmod{a_l}))$ 
            THEN
                2.2.11.1.1.  $d_i = d_i^{(a_k)}$ ,  $e_i = e_i^{(a_k)}$ ;
                2.2.11.1.2.  $k_1 = k_1/a_k$ ,  $k_2 = k_2/a_k$ ;
    2.3.  $i = i + 1$ ;
3. OUTPUT  $(a_1, a_2, \dots, a_J) NAF_{w,t}(k_1, k_2) = \begin{pmatrix} (d_1^{(a_1)}, d_2^{(a_2)}, \dots, d_l^{(a_l)}) \\ (e_1^{(a_1)}, e_2^{(a_2)}, \dots, e_l^{(a_l)}) \end{pmatrix}$ .

```

4. Performance analysis

We have studied the performance of our algorithm according to the generally accepted framework described in [2]. That is, we consider certain efficient representations of a group of points of an elliptic curve E over $GF(p)$, express complexity of basic operations in E , such as addition, doubling, tripling etc. in terms of multiplication in $GF(p)$, set a series of experiments to determine average characteristics of our representation and compute resulting complexity of the algorithm.

Average complexity of multi-scalar multiplication in $E(GF(p))$, $p \approx 2^{512}$

	$(2, 3) \overrightarrow{NAF}_6$	$(2, 3) \overleftarrow{NAF}_6$	$(2, 3) NAF_{6,8}$	$(2, 3) NAF_{6,15}$
Jacobian coordinates	6117.5	6115.7	6405	6136.1
Inverted Edwards	4661.8	4986	4832.8	4659.9
Inverted twisted Edwards	4639.5	4736	4837.6	4657.6

No wonder that our algorithm for a window w behaves generally like $\overrightarrow{NAF}_w$. At the same time we are able to reduce memory costs twice, while performance loss is about 4%.

We conclude that the algorithm presented has nice performance characteristics, allows to reduce memory cost and is, in general, more suited for implementation than the original variants of joint multibase NAF.

References

- [1] Doche C., Kohel D. R., Sica F., “Double-base number systems for multi-scalar multiplications”. In: “*Advances in Cryptology—EUROCRYPT 2009*”, Lect. Notes Comput. Sci., **5479** 2009, 502–517.
- [2] Dygin D. M., Grebnev S. V., “Efficient implementation of the GOST R 34.10 digital signature scheme using modern approaches to elliptic curve scalar multiplication”, *Математические вопросы криптографии (Math. Aspects Cryptogr.)*, **4:2** (2013), 47–57.
- [3] Bernstein D. J., Lange T., *Explicit-Formulas Database*, 2012, <http://hyperelliptic.org/EFD>.
- [4] Hişil H., *Elliptic Curves, Group Law and Efficient Computation*, Ph. D. thesis, Brisbane: Inform. Secur. Inst., Queensland Univ. Technology, 2010, xx+225 pp., <http://eprints.qut.edu.au/33233/>.
- [5] Kalinin M., *An Implementation of the GOST R 34.10 Digital Signature Scheme Using Novel Algorithms of Elliptic Curve Scalar Multiplication*, Master thesis, Moscow: Lomonosov State Univ., 2010 (In Russian), 50 pp.
- [6] Longa P., *Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields*, Master Sci. thesis, Ottawa: Univ. Ottawa, 2007, xv+173 pp., <http://eprint.iacr.org/2008/100>.
- [7] Longa P., Gebotys C., *Setting Speed Records with the (Fractional) Multibase Non-Adjacent Form Method for Efficient Elliptic Curve Scalar Multiplication*, CACR Tech. Rep., Waterloo: Centre Appl. Cryptogr. Res., Univ. Waterloo, 2008, 40 pp., <https://eprint.iacr.org/2008/118.pdf>.
- [8] Longa P., Miri A., “New multibase non-adjacent form scalar multiplication and its application to elliptic curve cryptosystems” (2008), 39 pp., <http://eprint.iacr.org/2008/052>.
- [9] Solinas J. A., *Low-Weight Binary Representations for Pairs of Integers*, Waterloo: Univ. Waterloo, 2001, <http://cacr.uwaterloo.ca/>.