On-the-Fly Multi-Base Recoding for ECC Scalar Multiplication without Pre-Computations

Thomas Chabrier, Arnaud Tisserand

On-the-Fly Multi-Base Recoding for ECC Scalar Multiplication without Pre-Computations

Thomas Chabrier[3,1] and Arnaud Tisserand[2,1]
IRISA[1], CNRS[2], University Rennes 1[3], INRIA Centre Rennes - Bretagne Atlantique
6 rue Kerampont, CS 80518, 22305 Lannion cedex, FRANCE
arnaud.tisserand@irisa.fr

***Abstract*—Scalar recoding is popular to speed up ECC scalar multiplication: non-adjacent form, double-base number system, multi-base number system. But fast recoding methods require pre-computations: multiples of base point or off-line conversion. In this paper, we present a multi-base recoding method for ECC scalar multiplication based on i) a greedy algorithm starting least significant terms first, ii) cheap divisibility tests by multi-base elements and iii) fast exact divisions by multi-base elements. Multi-base terms are obtained on-the-fly using a special recoding unit which operates in parallel to curve-level operations and at very high speed. This ensures that all recoding steps are performed fast enough to schedule the next curve-level operations without interruptions. The proposed method can be fully implemented in hardware without pre-computations. We report FPGA implementation details and very good performances compared to state-of-art results.**

***Index Terms*—elliptic curve cryptography; scalar multiplication; DBNS; MBNS; divisibility test; exact division by constant;**

I. Introduction

Scalar multiplication is the most time consuming operation in elliptic curve cryptography (ECC) protocols. It is denoted by $[k]P$ where P is a curve point and k a scalar. Basic scalar multiplication algorithm scans each bit of k and performs some curve-level operations depending on the bit value. Scalar representation significantly impacts the number of point operations to be executed and overall computation time. Consequently scalar recoding methods are very popular: non-adjacent forms (NAF and wNAF), double- or multi-base number systems (DBNS/MBNS), etc. Sec. II recalls these methods and basic ECC elements. Previous fast recoding methods require a pre-computation step prior to scalar multiplication. For wNAF, several multiples of P have to be precomputed and stored. For DBNS/MBNS, the scalar must be recoded off-line.

Below we present a method and its FPGA implementation to recode on-the-fly the scalar using MBNS without pre-computations. Our recoding is performed in parallel to curve-level operations. It uses very cheap divisibility tests for each base element and an efficient implementation of exact division algorithms used for multiple-precision arithmetic. Exact division refers to division where the remainder is known to be zero. Due to paper length limit, we only deal here with curves defined over $\mathbb{F}_p$ but our method can be easily applied in $\mathbb{F}_{2^m}$ case. Sec. III and IV present respectively unsigned and signed versions of our method. Section V compares our results to state-of-art ones.

1: $Q \leftarrow \mathcal{O}$
2: **for** i **from** $n-1$ **to** 0 **do**
3: $\quad Q \leftarrow 2Q$ (DBL)
4: $\quad$ **if** $k_i = 1$ **then** $Q \leftarrow Q + P$ (ADD)
5: **return** Q

Fig. 1. Double-and-add algorithm for scalar multiplication $Q = [k]P$

II. State-of-Art in ECC Scalar Multiplication

A brief introduction is presented below. The reader is referred to [1], [2] for further details. An elliptic curve E over the prime field $\mathbb{F}_p$, of large characteristic, can be defined by the simplified Weierstrass equation $y^2 = x^3+ax+b$ with *curve parameters* $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$. The rational points on the curve and a special point, called *point at infinity* denoted by $\mathcal{O}$, form an *abelian group* (denoted additively where $\mathcal{O}$ acts as the identity) on top of which the cryptosystem works. Given points P, Q on the curve, curve-level operations are defined: *point addition* $P + Q$ where $P \neq \pm Q$ (denoted by ADD) and *point doubling* $[2]P = P+P$ (DBL). *Scalar multiplication* $[k]P$ is defined by $[k]P = P+P+\cdots+P$ with $k-1$ additions. The scalar k is $(k_{n-1}k_{n-2}\dots k_1k_0)_2$ with n in the range 160–520 bits for typical cryptographic sizes.

Each *operation at curve-level* involves a sequence of *operations at field-level* (multiplication: M, square: S, inversion: I). Curve points can be represented using *affine coordinates* ($\mathcal{A}$): (x, y). In that case, ADD and DBL operations require expensive field inversions (in $\mathbb{F}_p$ one inversion is about 15 to 30 multiplications). Hence most efficient implementations use *projective coordinates*. In this paper, we use Jacobian coordinates ($\mathcal{J}$) as a popular class of projective coordinates where $(X : Y : Z)$ corresponds to the affine point $(X/Z^2, Y/Z^3)$ for $Z \neq 0$.

A. Basic Scalar Multiplication Methods

Basic scalar multiplication algorithm, called *double-and-add*, is presented on Fig. 1. Its average computation cost is $0.5n$ ADD $+ n$ DBL ($0.5n$ ones in k for security requirement).

Point addition at line 4 always uses the same point P. Then P can be kept in affine coordinates and used by *mixed addition* mADD ($\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$) in order to speed up the computation and reduce the P coordinates storage (see Sec. V-A for cost).

Point subtraction (SUB) is as efficient as point addition ($\mathcal{A}$: $-P = (x, -y)$ and $\mathcal{J}$: $-P = (X : -Y : Z)$ for curves

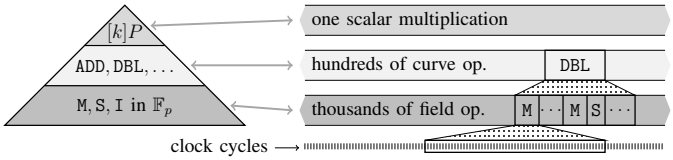

Fig. 2. Pyramid of operations in a scalar multiplication (arbitrary scale)

over $\mathbb{F}_p$). This motivates the use of signed digits such as NAF ($k_i \in \{0, \pm 1\}$) where no two consecutive signed digits are non-zero [1, Sec. 3.3.1]. Scalar multiplication using NAF recoding is straightforward: replace line 4 in Fig. 1 by "**if** $k_i \neq 0$ **then** $Q \leftarrow Q$ sign(k_i) P". The average computation cost is $0.3n$ ADD $+ n$ DBL (cost for SUB is the same than ADD).

Another optimization, called wNAF, processes a window of w digits of k at a time. wNAF uses digits $k_i \in \{0, \pm 1, \pm 3, \pm 5, \ldots, \pm 2^{w-1} - 1\}$, and at most one of any w consecutive digits is non-zero [1, Sec. 3.3.1]. Multiples $P_j = [j]P$ have to be pre-computed and stored for all $j \in \{3, 5, \ldots, 2^{w-1} - 1\}$. Scalar multiplication is done using ADD/SUB of pre-computed multiple P_j (corresponding pseudo-code is "**if** $k_i \neq 0$ **then** $Q \leftarrow Q$ sign(k_i) $P_{|k_i|}$"). The average computation cost is $n/(w+1)$ ADD $+ n$ DBL without the pre-computation step. These pre-computations may be interesting if the same point P is reused. In practice, wNAF is used with $w \leq 4$ for limited storage overhead.

Fig. 2 illustrates the typical number of operations required at each level. One $[k]P$ operation requires hundreds of curve-level operations. Each curve operation (ADD, DBL) requires a sequence of 8–12 field-level operations. Finally, each field operation requires tens (for large operators) to hundreds (for small iterative operators) of clock cycles.

B. Scalar Multiplication using Double-Base Number System

DBNS was initially introduced in [3], used for modular exponentiation in [4], for signal processing in [5] and for ECC in [6], [7], [8], [9], [10] and [11] (which is very complete). In DBNS, number x is represented by the sum of mixed powers of two co-prime integers b_1 and b_2, the two bases, typically $(b_1, b_2) = (2, 3)$, such that $x = \sum_{i=1}^{n'} x_i b_1^{u_i} b_2^{v_i}$ with $x_i = \pm 1$. Unsigned DBNS ($x_i = 1$) leads to larger terms number n'.

For ECC computations in DBNS, a new curve-level operation has to be defined: *point tripling* $[3]P = P + P + P$ (denoted TPL). It is faster than ADD (see Sec. V-A for cost). DBNS is a very sparse representation (number of terms n' is very small compared to number of bits n in standard binary representation). Then, the number of point additions is reduced. In [7], a special type of DBNS recoding, called DBNS chain, is proposed with an Horner like factorization of DBLs and TPLs operations (under conditions $u_1 \geq u_2 \geq \ldots \geq u_{n'}$ and $v_1 \geq v_2 \geq \ldots \geq v_{n'}$) leading to higher improvement. We will report performances of some DBNS scalar multiplication algorithms from literature in Sec. V. There are DBNS scalar multiplication extensions using pre-computed multiples of P leading to higher speed but with a higher storage cost [11].

DBNS helps to reduce the total computation time. But binary to DBNS conversion is performed off-line. Most of proposed conversions proceed most significant terms first by subtracting to k a good/best approximation of k by a term of form $2^u 3^v$ using huge tables or expensive computations. The authors from [10] claim that tree based approach conversion is too costly for hardware implementation of systems using integers in the cryptographic range (p. 437, Sec. 3). In [8], 10 to 72 points have to be pre-computed and stored (a better usage of silicon area should be a parallel architecture). In [12], an FPGA implementation of binary to DBNS conversion is proposed but only for very small operands ($n \leq 20$ bits) in signal processing applications.

DBNS is a very redundant number system. In [13], this redundancy is used to randomly select the recoding as a counter-measure against some side-channel attacks (SCAs).

C. Scalar Multiplication using Multi-Base Number System

MBNS is a generalization of DBNS with more than two bases [14], [15], [16], [17], [18], [19] and [20]. A multi-base $\mathcal{B}$ is a tuple of l co-prime integers $(b_1, b_2, \ldots, b_l)$. Number x is represented as the sum of terms $x = \sum_{i=1}^{n'} \left(x_i \prod_{j=1}^{l} b_j^{e_{j,i}}\right)$ with $x_i = \pm 1$. MBNS is a very sparse and redundant representation. In literature, proposed multi-bases are often $(2, 3, 5)$ and $(2, 3, 5, 7)$.

For ECC scalar multiplication in MBNS, new curve-level operations have to be defined: *point quintupling* $[5]P$ (QPL), *point septupling* $[7]P$ (SPL), *point eleventupling* $[11]P$ (EPL), etc. These new operations are more efficient than equivalent sequences of ADD, DBL and TPL operations (see Sec. V-A for typical costs). MBNS scalar multiplication is similar to DBNS algorithms with more curve-level operations QPL, SPL, etc.

MBNS suffers from the same limitation as DBNS: the need for off-line conversion with huge tables and/or long pre-computations. In [14] and [17] conversion uses good approximations of k using terms of form $\pm \prod_{j=1}^{l} b_j^{e_j}$ similarly to DBNS conversion. In [15] and [16] conversion uses an adaptation of wNAF with detection of b_j multiples into a limited window, but it requires pre-computations and additional storage. To our knowledge, [15] and [16] provide the best MBNS results but without hardware implementation details.

III. Proposed Method

Notations used in paper remainder are:

- $k = (k_{n-1} k_{n-2} \ldots k_1 k_0)_2$, $k > 1$, the n-bit *scalar* stored into t words of w bits with $w(t-1) < n \leq wt$ (i.e. last word may be 0-padded). $k^{(i)}$ the ith word of k starting from least significant for $0 \leq i < t$.
- $\mathcal{B}$ the *multi-base* with l *base elements* (co-prime integers), $\mathcal{B} = (b_1, b_2, \ldots, b_l)$.
- predicate divisible$(x, \mathcal{B})$ returns true if x is divisible by at least one base element in $\mathcal{B}$ (false for other cases).
- number x represented as the *sum of terms* $x = \sum_{i=1}^{n'} \left(d_i \prod_{j=1}^{l} b_j^{e_{j,i}}\right)$ with $d_i = \pm 1$.
- *term* $(d_i, e_{1,i}, e_{2,i}, \ldots, e_{l,i})$ defined by $d_i \times \prod_{j=1}^{l} b_j^{e_{j,i}}$ in $\mathcal{B}$ (index i may be omitted when context is clear).

1: LT $\leftarrow \emptyset$
2: **while** $k > 1$ **do**
3: **if** not$\big($ divisible$(k, \mathcal{B})\big)$ **then** (*divisibility test*)
4: $d \leftarrow 1$
5: $k \leftarrow k - 1$
6: **else**
7: $d \leftarrow 0$
8: **for** j **from** 1 **to** l **do**
9: $e_j \leftarrow 0$
10: **while** $k \equiv 0 \bmod b_j$ **do** (*divisibility test*)
11: $e_j \leftarrow e_j + 1$
12: $k \leftarrow k/b_j$ (*exact division*)
13: LT $\leftarrow$ LT $\cup\, (d, e_1, e_2, \ldots, e_l)$
14: **return** LT

Fig. 3. Unsigned MBNS recoding algorithm

- Q, P curve points and $Q = [k]P$ scalar multiplication.

Due to space limitation, we only present results for elliptic curves defined over $\mathbb{F}_p$, but it can be used for $\mathbb{F}_{2^m}$ (fine tuning is slightly different due to different cost ratios I/M and S/M). In this section, we present a simple unsigned version ($d_i = 1$) for the sake of simplicity. Sec. IV details algorithms for signed representations ($d_i = \pm 1$). Units described in this section can be directly used or slightly adapted for signed representation.

A. Unsigned Algorithms

Our MBNS unsigned recoding algorithm, see Fig. 3, is very simple. Divisibility of k by $\mathcal{B}$ elements is tested. When k is not divisible, 1 is subtracted to k. $b_1 = 2$ is selected for efficiency purpose (divisibility is ensured 50% of time). For lines 8–12, the scalar k is divided by all base elements b_j in $\mathcal{B}$ as much as possible using cheap divisibility tests and exact divisions. This division step provides the term exponents $e_1, e_2, \ldots, e_l$. LT denotes the list of terms which stores the MBNS recoding of k, LT $= \big((d_1, e_{1,1}, e_{2,1}, \ldots, e_{l,1}), (d_2, e_{1,2}, e_{2,2}, \ldots, e_{l,2}), \ldots\big)$ with $d_i \in \{0, 1\}$. Only the first term may have $d_1 = 0$ (if the initial k is immediately divisible in $\mathcal{B}$). Divisibility tests at line 3 and 10 can be shared. The algorithm stops when $k \leq 1$ due to Horner form such as $2^a 3^b 5^c\big(1 + 2^{a'} 3^{b'} 5^{c'}(1)\big)$.

Divisibility tests are detailed in Sec. III-B. There are implemented using $t + \varepsilon$ clock cycles (ε is a small constant) for all $b_j \neq 2$ and only one for $b_j = 2^s$ with $s \leq w$. Once k is divisible by b_j, we use fast exact division algorithms to perform $k \leftarrow k/b_j$ as detailed in Sec. III-C and with $t + \varepsilon'$ clock cycles for all $b_j \neq 2$.

MBNS recoding algorithm in Fig. 3 works in a *serial way*: one multi-base term at a time and starting with the least significant one. Each term can be immediately used in the scalar multiplication algorithm in Fig. 4. This algorithm computes $Q = [k]P$ using LT and is a multi-base adaptation of the standard left-to-right scalar multiplication algorithm (see for instance [1, Sec. 3.3.1]). Operation $Q + d \times P$ at line 3 is NOP (no operation) or ADD since $d \in \{0, 1\}$.

The combination of recoding (Fig. 3) and scalar multiplication (Fig. 4) algorithms allows to overlap recoding steps by curve-level operations. For instance, when divisibility by 3 is detected, exact division by 3 and TPL operations can

1: $Q \leftarrow \mathcal{O}$
2: **foreach** t **in** LT **do** $(t = (d, e_1, e_2, \ldots, e_l))$
3: $Q \leftarrow Q + d \times P$ $(d \in \{0, 1\} \Rightarrow$ NOP/ADD$)$
4: **for** j **from** 1 **to** l **do**
5: $P \leftarrow \big[b_j^{e_j}\big]P$ (DBL, TPL, QPL, ...)
6: $Q \leftarrow Q + P$
7: **return** Q

Fig. 4. MBNS scalar multiplication algorithm $Q = [k]P$

be launched in parallel. Our recoding algorithm produces a MBNS representation with a recursive factorization similar to Horner scheme. Fig. 10 illustrates a complete example. Unlike previous DBNS and MBNS recoding methods, ours can be fully embedded in hardware and operates on-the-fly. First, we do not need costly tables or computations such as the approximation of k by multi-base terms. Second, as soon as a divisibility is detected, we can launch the corresponding curve-level operation.

As we start with least significant terms first, we cannot use mixed coordinates point addition (mADD). We are obliged to use standard point addition which is a little slower. Clearly our method is not competitive compared to the fastest state-of-art ones when costly off-line recoding is possible. But it provides the first full on-the-fly hardware implementation.

Overlapping recoding operations by curve-level operations is possible due to the very fast divisibility tests and exact divisions. For instance, with $n = 160$, $w = 12$ and $t = 14$, divisibility tests by all $b_j \neq 2$ and exact division by one $b_j \neq 2$ respectively require $t + 3 = 17$ and $t + 4 = 18$ clock cycles. These small durations have to be compared to the duration of one DBL, TPL, QPL, etc. which are significantly slower.

There is a short latency at the very beginning (less than 0.01% of total $[k]P$ computation time for $n = 160$ and even less for larger fields). The first curve-level operation is determined using the first divisibility test results. After $t + \varepsilon$ clock cycles there are two cases: i) k is divisible by one multi-base element b_j then a DBL, TPL, QPL, etc. can be launched depending on which b_j divides k, ii) k is not divisible by $\mathcal{B}$ elements and an ADD can be launched.

In this work we do not study an analytical evaluation of the number of each type of curve-level operation. But we provide extensive experimental evaluations of both these types of operations and complete scalar multiplication duration.

Selection of $\mathcal{B}$ elements is required. Fig. 5 reports statistical $[k]P$ timings (in M) for 10 000 random 160-bit values recoded using our unsigned MBNS algorithm for various multi-bases.

Scalar multiplication algorithm in Fig. 4 is not secure

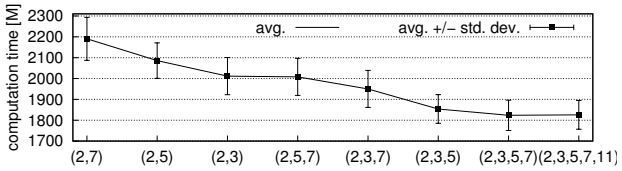

Fig. 5. Statistical timings of unsigned MBNS scalar multiplication

b_j	i											
	11	10	9	8	7	6	5	4	3	2	1	0
3	2	1	2	1	2	1	2	1	2	1	**2**	**1**
5	3	4	2	1	3	4	2	1	**3**	**4**	**2**	**1**
7	4	2	1	4	2	1	4	2	1	**4**	**2**	**1**

TABLE I
PASCAL'S TAPES FOR DIVISIBILITY TESTS (VALUES ARE $2^i \bmod b_j$)

against SCAs. Simple power analysis attacks can be used due to the different behavior of curve-level operations in lines 3 and 5. We will see in Sec. IV how signed MBNS recoding can be used as a protection against some attacks.

B. Implementation of the Divisibility Tests

At each recoding step, the scalar remainder has to be tested for divisibility by all b_j for $1 \leq j \leq l$. Testing divisibility by 2^s with $s \leq w$ in a radix-2 representation is straightforward and implemented in a very small module of the recoding unit, see Sec. III-D. For $b_j \neq 2$, we use a very old method based on specific properties of the sum of argument digits modulo b_j. This method for divisibility test by b_j in radix-r representation is reported in a Blaise Pascal's post-mortem publication [21] (in Latin, see [22] for comments in English). This method is often called Pascal's tape. We only provide details for $b_j \in \{3, 5, 7\}$ as they lead to the most efficient $\mathcal{B}$. Tab. I reports the remainders $2^i \bmod b_j$. They form a periodic sequence.

Using Tab. I with Pascal's tape in case $b_j = 3$, the periodic sequence is $(2\,1)^*$, then one has:

$$\begin{aligned} k \bmod 3 &= (\ldots + 2^3 k_3 + 2^2 k_2 + 2^1 k_1 + k_0) \bmod 3 \\ &= (\ldots + 2k_3 + k_2 + 2k_1 + k_0) \bmod 3 \\ &= \Big(\underbrace{\sum (2k_{2i+1} + k_{2i})}_{\alpha}\Big) \bmod 3. \end{aligned}$$

Computation of α requires the sum of many 2-bit words ($n \gg 100$). α is a multi-bit integer, then it has to be recursively reduced using the same method. There is a trade-off between the size of the intermediate accumulators and the reduction completion. Architecture presented on Fig. 6 decomposes this large operation into partial sums accumulated and partial reduction on a limited number of bits for each word of the scalar. This is the purpose of the light block denoted "$\sum$ `for` $b_j = 3$" and connected register on Fig. 6. Then t cycles are required for the accumulation and partial reduction. The very last reduction steps and comparison to b_j is denoted "$\mathcal{R}$ `for` b_j" in the second type of light block. Clock, reset and enable signals are not represented on the figures.

For $b_j = 7$, the sum uses 3-bit words with the sequence $(4\,2\,1)^*$. But for $b_j = 5$, the sequence is $(3\,4\,2\,1)^*$ where digit 3 requires a specific treatment. A first solution uses $3 = 1 + 2$ and an unsigned sum with additional inputs. A second solution considers $3 \equiv -2 \bmod 5$ and a signed sum with less operands (but with sign extension). Architecture on Fig. 6 allows parallel divisibility test by several b_j in only one computation.

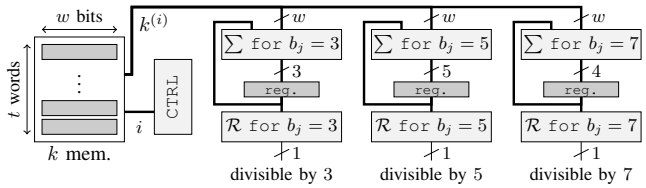

Fig. 6. Divisibility tests architecture

w	area slices (FF/LUT)	freq. MHz	clock cycles
12	25 (40/81)	543	$t + 3$
24	67 (53/152)	549	$t + 4$

TABLE II
FPGA IMPLEMENTATION RESULTS FOR DIVISIBILITY TESTS

Parameter w significantly impacts performances. The lengths of remainders sequences for $b_j = 3, 5, 7$ are respectively $2, 4, 3$ (from Tab. I). To avoid complex decoding schemes, we use $w = \mathrm{lcm}(2, 4, 3) = 12$ and $w = 24$ (larger multiples of 12 slow down the recoding unit).

Hardware implementations reported below have been described in VHDL and implemented on a XC5VLX50T FPGA using ISE 12.4 from Xilinx with standard efforts for synthesis, place and route. We report numbers of clock cycles, best clock frequencies and numbers of occupied slices. We also report numbers of look-up tables (LUTs with 6 inputs in Virtex 5) and flip-flops (FFs) for area. A XC5VLX50T contains 7 200 slices with 4 LUT and 4 flip-flops per slice. We use flip-flops for all storage elements. FPGA implementation results are reported in Tab. II for divisibility tests by $b_j \in \{3, 5, 7\}$ and $n = 160$.

C. Implementation of the Exact Division by b_j Elements

Here, *exact division* k/b_j means that we know that dividend k is divisible by divisor b_j (using divisibility tests presented above). This significantly optimizes the division. [23] provides an efficient algorithm when the radix is prime or power of 2. Division by 2^s is straightforward and is not considered in this section (a shifter is included in the recoding unit Sec. III-D).

We use a dedicated version of algorithm presented in [23] for $b_j \in \{3, 5, 7\}$ and optimized for hardware implementation. Our algorithm, presented in Fig. 7, operates in t iterations in a word-serial way starting with least significant. Iteration number i deals with $k^{(i)}$ the ith word of k. The inverse of divisor b_j modulo 2^w is a constant and always exists since multi-base elements are co-prime and include 2.

The main differences for the 3 operations ($b_j \in \{3, 5, 7\}$) are the multiplication by modular inverse on line 4 and comparisons to constants on line 7. All other elements are shared in the architecture (operators, control, registers).

Tab. III reports binary representations of modular inverses for exact division by 3, 5, 7. Multiplication $r \times (b_j^{-1} \bmod 2^w)$ at line 4 in Fig. 7 is implemented as a sequence of additions/subtractions and shifts using an in-house multiplication by constants algorithm [24]. Subtraction at line 3 is

b_j	$b_j^{-1} \bmod 2^{12}, \gamma$	$b_j^{-1} \bmod 2^{24}, \gamma$
3	$(101010101011)_2$, 3	$(101010101010101010101011)_2$, 4
5	$(110011001101)_2$, 3	$(110011001100110011001101)_2$, 4
7	$(110110110111)_2$, 3	$(110110110110110110110111)_2$, 4

TABLE III
MODULAR INVERSES USED IN EXACT DIVISIONS

w	area slices (FF/LUT)	freq. MHz	clock cycles
12	59 (138/171)	291	$t+4$
24	152 (441/448)	202	$t+5$

TABLE IV
FPGA IMPLEMENTATION RESULTS FOR EXACT DIVISION.

inserted in the sequence. Some adders in the 3 sequences are shared to reduce area. Tab. III reports γ the number of additions/subtractions required to perform $r \times (b_j^{-1} \bmod 2^w)$.

The architecture of our exact division by $b_j \in \{3,5,7\}$ unit is presented on Fig. 8. At iteration i, word $k^{(i)}$, read (R port) from scalar memory, is added to $-c$ and used in the addition sequence (block denoted "$\times(b_j \bmod 2^w)$" and "$\pm$ seq.") corresponding to b_j. The correct value is selected by MUX1 and written in scalar memory (W port). We use an in-place version of the algorithm $k \leftarrow k/b_j$ to keep memory footprint as small as possible. Comparisons in loop lines 6–8 of algorithm Fig. 7 are unrolled and implemented as combinatorial logic ("cmp. b_j" block) for each $b_j \in \{3,5,7\}$. The correct value c is selected by MUX2 and sent the addition sequences. The FPGA implementation results for our exact division unit are reported in Tab. IV for $n = 160$. For $w = 24$, speed decreases due to the complexity of the comparison blocks. For example, in case $b_j = 7$, six comparisons are required.

D. Unsigned Multiple-Base Recoding Unit

The complete recoding unit architecture is presented on Fig. 9. The scalar memory stores k. The small subtraction block is in charge of line 5 in the recoding algorithm Fig. 3. The DTD-2 block is in charge of divisibility test (1-bit result) and exact division (w-bit bus) by 2 or small powers of 2 (2^s with $s \leq w$, if $s > w$ several iterations are used). Divisibility test unit for other base elements is described in Sec. III-B (3-bit output) while the exact division unit is

1: $c \leftarrow 0$
2: **for** i **from** 0 **to** $t-1$ **do**
3: $\quad r \leftarrow k^{(i)} - c$
4: $\quad r \leftarrow r \times (b_j^{-1} \bmod 2^w)$
5: $\quad c \leftarrow 0$
6: $\quad$ **for** h **from** 1 **to** $b_j - 1$ **do**
7: $\quad\quad$ **if** $r \geq h \times \lceil (2^w - 1)/b_j \rceil$ **then**
8: $\quad\quad\quad c \leftarrow c + 1$
9: $\quad k^{(i)} \leftarrow (r_{w-1} \cdots r_0)$
10: **return** k

Fig. 7. Exact division algorithm for k/b_j

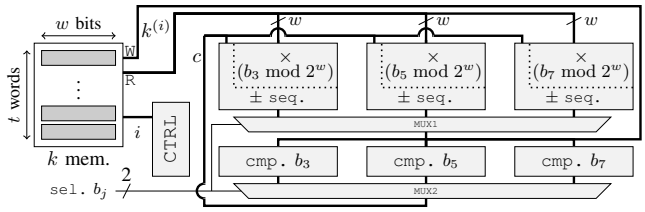

Fig. 8. Exact division unit architecture

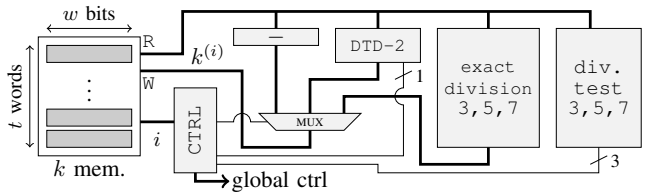

Fig. 9. Complete recoding unit architecture

described in Sec. III-C (w-bit output). MUX selects which unit output should be written in the scalar memory. The global controller (CTRL) generates all control signals for units. It also provides the global control with informations on which curve-level operations have to be launched.

FPGA implementation results for the complete unsigned recoding unit are reported in Tab. V for $\mathcal{B} = (2,3,5,7)$ and $n = 160$ (see Sec. IV-B for comparison to a ECC processor).

Our recoding unit operates significantly faster than curve-level operations and in parallel to them. It provides on-the-fly informations on which curve-level operations to be launched without interruptions as illustrated on Fig. 10. On this figure, "CLO" denotes curve-level operations, DT denotes divisibility test, "res." their results and "$/b_j$" exact division by b_j. For $k = 87$, the recoding is $87 = 0 + 3^1 \times (1 + 2^2 7^1)$.

The first term $(0 + 3)$ recoding is performed as fast as possible while the second one $(1 + 2^2 7^1)$ is spread over the computations curve-level operations, without interruption. This provides us options when designing the control.

For $n = 160$, $w = 12$ and $t = 14$, recoding a scalar corresponding to $2^8 3^4 5^2 7^1$ requires 292 clock cycles. This is less than one DBL operation (even for a parallel architecture). It reduces to 195 clock cycles for $w = 24$ and $t = 7$. The same term recoding requires 772 clocks cycles for $n = 521$ and $w = 12$ (435 for $w = 24$).

E. Validation

Both recoding and scalar multiplication algorithms have been implemented in PARI/GP (http://pari.math.u-bordeaux.

w	area slices (FF/LUT)	freq. MHz
12	153 (301/412)	232
24	323 (682/908)	202

TABLE V
FPGA IMPLEMENTATION RESULTS FOR COMPLETE RECODING UNIT

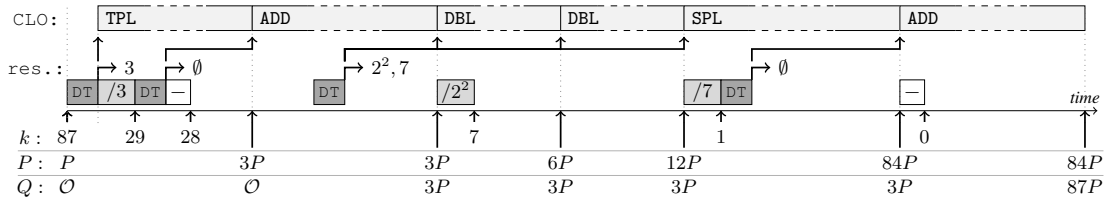

Fig. 10. MBNS recoding and scalar multiplication illustration for $k = 87$

unsigned version			*signed version*	
4:	$d \leftarrow 1$	$\longrightarrow$	4:	$d \leftarrow \mathrm{S}(k)$
5:	$k \leftarrow k - 1$		5:	$k \leftarrow k - d$

Fig. 11. Modifications between the unsigned and signed recoding algorithms

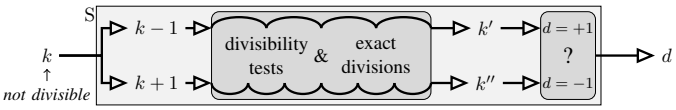

Fig. 12. Principle of the `min` selection function

fr/) and SAGE (http://www.sagemath.org/) mathematical softwares (each author was in charge of one version GP or SAGE). The results of the two versions have been compared to the mathematical values and cross-checked for very large random data sets (millions of scalars for many sizes $n \in [160, 520]$ bits). Functional validation of the architecture was done using some VHDL simulations on limited sets of random data and compared to the mathematical values. For performance validation, a lot of random tests and comparisons to state-of-art results have been performed (see Sec. V).

IV. Signed-Digit Optimizations

The unsigned recoding algorithm, Fig. 3 of Sec. III-A, only performs $k \leftarrow k-1$ when k is not divisible by $\mathcal{B}$ elements. As with other number systems, such as Booth recoding, wNAF, Avizienis representations, or DBNS, using *signed digits* may help us to reduce the number of terms.

A. Signed-Digit MBNS Recoding

A simple modification of our MBNS recoding algorithm Fig. 3 is required to support signed digits as illustrated on Fig. 11. A *selection function* S has been introduced to select the digit $d = \pm 1$ to be used for each term $(d, e_1, e_2, \ldots, e_l)$ when k is not divisible by $\mathcal{B}$ elements. Depending on d, updating the scalar requires a subtraction ($d = 1$ similarly to the unsigned version) or an addition ($d = -1$). The scalar multiplication algorithm Fig. 4 is unchanged. At curve level, digit values correspond to: a point addition when $d = 1$, a point subtraction when $d = -1$ and no operation when $d = 0$ (for the very first term only).

Determining S such that the recoding algorithm always produces the shortest list of terms is a very hard problem. We compared 4 heuristic selection functions and trade-offs between recoding performances (LT length) and implementation complexity (i.e. silicon area and recoding speed).

1) Minimum value selection function (`min`): `min` is illustrated on Fig. 12. When k is not divisible, then $k - 1$ and $k + 1$ will be divisible by, at least, 2 (we always use $b_1 = 2$ in practice) and potentially other b_j. For each value $k - 1$ and $k+1$, divisibility tests and exact division unit are used to produce k' and k''. k' (resp. k'') corresponds to $k - 1$ (resp. $k + 1$) divided as much as possible by $\mathcal{B}$ elements. S returns $d = 1$ if $k' < k''$, else it returns $d = -1$ (test $k' \leq k''$ leads to similar performances). The `min` selection function only provides a local minimum for the total number of terms.

A second scalar memory (t words $\times$ w bits) has been added in the recoding unit to store both k' and k''. The exponents corresponding to both $k - 1$ and $k + 1$ have been computed and stored during the exploration. The controller is adapted such that the correct set of exponents is selected.

2) Maximum number of divisors selection function (`max_nb_div`): In `min`, half of divisibility tests and exact divisions are discarded. In `max_nb_div`, the number of base elements b_j which divide $k - 1$ and $k + 1$ is counted. S returns d according to the maximum number of divisors among $k - 1$ and $k + 1$. Only divisibility unit for $k - 1$ and $k + 1$ is used, not the exact division unit. `max_nb_div` is a cheap optimization but with low efficiency (see Sec. IV-C).

3) Approximated minimum value selection function (`approx`): In order to provide a cheap optimization but with higher performances, the `approx` selection function compares approximations of k' and k'' from `min` (instead of computing them exactly like in `min`). Exponents e'_j (resp. e''_j) correspond to the divisibility test results for $k - 1$ (resp. $k + 1$). The approximations of k' and k'' are respectively defined by $\delta' = \lfloor \log_2(k-1) \rfloor + 1 - \sum_{j=1}^{l} e'_j \log_2(b_j)$ and $\delta'' = \lfloor \log_2(k+1) \rfloor + 1 - \sum_{j=1}^{l} e''_j \log_2(b_j)$ where $\lfloor \log_2(k-1) \rfloor + 1$ is the position of the most significant bit (MSB) of $k - 1$ (idem for $k + 1$). MSB positions can be easily detected using our divisibility test unit (during the t iterations loop for each word). Approximation of weight $\prod_{j=1}^{l} b_j^{e'_j}$ (or with e''_j exponents) uses the sum of multiplications by $\log_2(b_j)$ constants. Our divisibility unit returns limited exponents: $e'_j \leq 1$ for $b_j \neq 2$ and $e'_j \leq w$ for $b_j = 2$ (see Sec. III-B). Then, there is no need for multiplications. As an example, for $\mathcal{B} = (2, 3, 5, 7)$ $\delta' = \mathrm{MSB(k-1)} - e_{b_1=2} - 1.5e_{b_2=3} - 2.25e_{b_3=5} - 2.75e_{b_4=7}$ where $e_{b_1=2} \leq w$ and $e_{b_2=3}, e_{b_3=5}, e_{b_4=7} \leq 1$. The constants come from: $\log_2 3 \approx 1.59$, $\log_2 5 \approx 2.32$, and $\log_2 7 \approx 2.81$.

w	area slices (FF/LUT)	freq. MHz
12	173 (326/466)	232
24	345 (724/1 005)	202

TABLE VI
FPGA IMPLEMENTATION RESULTS FOR SIGNED RECODING UNIT.

version	memory type	area slices (FF/LUT)	area BRAM	freq. MHz
small	distributed	2 204 (3 971/6 816)	0	155
	BRAM	1 793 (3 641/6 182)	6	155
large	distributed	3 182 (4 668/7 361)	0	142
	BRAM	2 427 (4 297/6 981)	6	142

TABLE VII
FPGA IMPLEMENTATION RESULTS FOR A COMPLETE ECC PROCESSOR OVER $\mathbb{F}_p$ WITH $n = 160$ BITS (FROM [25])

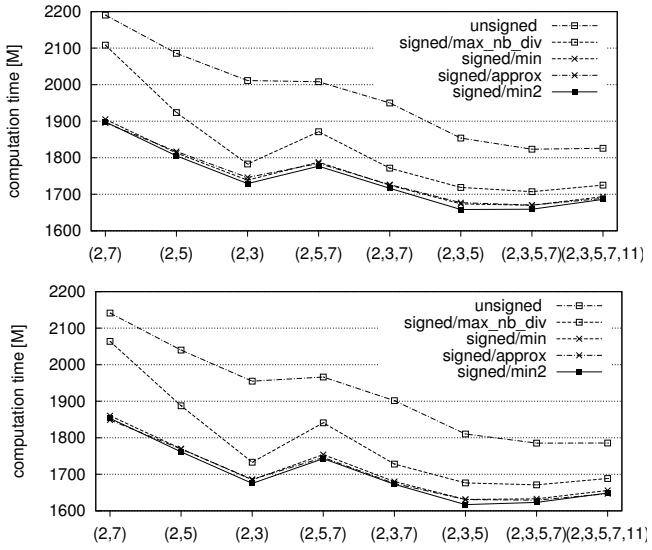

Fig. 13. Statistical performance evaluation of signed MBNS recoding and scalar multiplication (top: $a \neq -3$, bottom: $a = -3$)

The approximation for both k' and k'', as well as their comparison, can be easily implemented using a very small circuit (see [24]).

4) 2 steps minimum value selection function (`min2`): It uses the time margin illustrated on Fig. 10 using a recursion limited to the next term. The first step uses `min` with $(k-1, k+1)$ to produce (k', k''). The second step uses `min` with $(k'-1, k'+1, k''-1, k''+1)$ to produce $(\zeta', \zeta'', \zeta''', \zeta'''')$. S returns d according to the minimum value among $\zeta', \zeta'', \zeta''', \zeta''''$.

B. FPGA Implementation

Signed MBNS recoding unit has been implemented on FPGA (see end of Sec. III-B for target and tools details). Tab. VI reports corresponding results for `approx` selection function, $\mathcal{B} = (2, 3, 5, 7)$ and $n = 160$ bits. Compared to the unsigned version Tab. V, area overhead for signed version is very small: 13% more slices (8% FF and 13% LUTs). In Virtex 5 FPGAs, very small memories, such scalar ones, can be efficiently implemented using distributed RAMs in the LUTs of SLICEMs. This explains why only 25 additional flip-flops are required for the signed version while there is a 168-bit memory ($t = 14$ and $w = 12$) for the second scalar memory. The same speed is achieved for both signed and unsigned versions (the critical path is in the exact division unit).

In order to compare our MBNS recoding unit to a complete ECC processor, Tab. VII reports two FPGA implementations of an ECC processor provided by the authors of [25] (for curves over $\mathbb{F}_p$, $n = 160$ bits and Jacobian coordinates). The first one (small version) uses NAF method with one arithmetic unit per field operation, while the second (large version) uses 4NAF method and two arithmetic units per field operation but one modular inversion. Our MBNS signed recoding unit work at higher frequency than the ECC processor. And its area represents less than 10% (resp. 7%) for $w = 12$ compared to the complete small (resp. large) version of the ECC processor.

C. Experimental Analysis

Fig. 13 presents statistical analysis results for average computation time (in `M`) of 10 000 scalar multiplications with 160-bit random scalars recoded using our signed MBNS algorithm for various multi-bases and the 4 selection functions presented in Sec. IV-A. Most efficient multi-bases are $\mathcal{B} = (2, 3, 5)$ and $\mathcal{B} = (2, 3, 5, 7)$ with very close performances. Selection function `max_nb_div` is not efficient, `min` and `approx` have very close performances, and `approx` is the best trade-off between performances and recoding cost (`min` requires longer computations and more energy).

D. Randomized Selection Function

Side-channel attacks exploit some correlations between secret values manipulated in the device and physical parameters measured on the device such as power consumption, electromagnetic emanations or computation timing. Refer to [26] for a complete introduction on power analysis based SCAs. Typical SCAs and counter-measures for ECC are summarized respectively in [27] and [28]. Protections against simple attacks (based on only one or a very few traces) mainly use *uniformization* (or atomicity) and *randomization* methods. Protections against differential attacks (based on statistics over many traces) mainly use randomization methods.

We experimented with a simple randomized selection function (`rnd`) as a protection against some SCAs. When k is not divisible by $\mathcal{B}$ elements, S returns $d = 1$ or $d = -1$ randomly. Obviously this leads to larger number of terms (and point additions) in the recoding as reported in Tab. VIII (for simplified Weierstrass curves with $a \neq -3$ and 10 000 random scalars). Proposed randomization scheme allows a scalar substring to be represented using totally different recodings. This is a direct protection against some differential attacks due to the very huge number of different representations using signed digits [14, Tab. 2]. For protection against simple attacks, real

$\mathcal{B}$	rnd		min		diff.
	M	#ADD	M	#ADD	[%]
(2,3)	1 960.5	49.3	1 738.5	34.0	12.8
(2,3,5)	1 843.0	39.8	1 673.7	28.0	10.1
(2,3,5,7)	1 811.4	34.8	1 670.0	24.8	8.5
(2,3,5,7,11)	1 816.7	32.1	1 693.5	22.9	7.3

TABLE VIII
AVERAGE COMPUTATION TIME (M) AND POINT ADDITION NUMBER (#ADD) USING THE RANDOMIZED SELECTION FUNCTION ($a \neq -3$)

robustness of our randomized selection function relies on the fact that point addition and point subtraction cannot be distinguished in traces. Indeed, protecting the sign change when using point subtraction is supposed to be simple in the circuit. But we still have to perform a more complete security evaluation at hardware level using a real attack system and to compare to other protection schemes (e.g. addition chains [29]).

V. Comparison to State-of-Art

Below we compare our MBNS recoding and scalar multiplication algorithms for various multi-bases to state-of-art methods. We report results over $\mathbb{F}_p$ for simplified Weierstrass curves with unspecified parameter a and $a = -3$, using Jacobian coordinates, similarly to most of DBNS/MBNS references.

A. Costs of Curve-Level Operations

Tab. IX reports best computation costs, given in field-level operations (M, S) for various curve-level operations over $\mathbb{F}_p$ from literature. EFD is the excellent web site Explicit-Formulas Database http://hyperelliptic.org/EFD. We apply the typical cost assumption used in many references: $1\mathtt{S} = 0.8\,\mathtt{M}$. λDBL (resp. λTPL) denotes a sequence of λ successive DBL (resp. TPL) operations (e.g. $k = 2^\lambda$ or $k = 3^\lambda$). We use λDBL or λTPL operations when they are faster than their equivalent sequence of DBL or TPL.

B. Performance Comparisons

Some previous scalar multiplication algorithms require additional points to speed up computations. These additional points are multiples of the initial point P and stored in the cryptoprocessor during the complete scalar multiplication (2 n-bit registers per additional point). Most of methods assume pre-computed points represented using affine coordinates to benefit from fast mixed coordinates addition mADD. Tab. X reports costs of typical pre-computations. Costs at field-level include a conversion to affine coordinates which requires field inversions (usually 1M+1S+2I per addition point). We assume $1\mathtt{I} = 15\mathtt{M}$ for $\mathbb{F}_p$ inversion.

These costs can be neglected for multiple successive $[k]P$ operations with the same P, but it is not the case if P changes before each scalar multiplication (e.g. support of various protocols/sizes, base point randomization method, ...).

Tab. XI and XII compares scalar multiplication methods from literature to our signed MBNS method using 10 000 random scalars and approx selection function. In these tables, DBNS results have been computed using the PARI/GP program kindly provided by the author of [7]. This program generates a signed DBNS chain using pre-computations and where approximations are obtained by a search table.

The results presented in references [16] and [32] are based on a left-to-right MBNS scalar multiplication algorithm to benefit from mADD while the scalar is recoded using a right-to-left algorithm (this strategy prevents them from providing an on-the-fly computation). If we use a similar strategy, the computation cost reduction is estimated to $\big((11 + 5 \times 0.8) - (7 + 4 \times 0.8)\big)$ times the number of ADD operations. In case $\mathcal{B} = (2, 3, 5)$ and $n = 160$, this leads to a reduction about 134M. Hence, references [16] and [32] are still faster than our method but with a much smaller difference.

VI. Conclusion

In this paper, we proposed a simple multi-base recoding algorithm for ECC scalar multiplication in hardware without any pre-computations. The scalar recoding is performed on-the-fly and in parallel to curve-level operations without additional latency. The proposed recoding circuit uses cheap divisibility test by multi-base elements and exact division using very small dedicated hardware units. Our MBNS recoding and scalar multiplication method is a little less competitive compared to other DBNS/MBNS methods when pre-computations or off-line recoding can be used. But our method leads to more efficient solutions in embedded applications fully integrated in hardware without resources for costly recoding and limited storage. As future work, we plan to deal with more advanced recoding schemes to reduce the number of produced terms and improved randomization schemes to increase robustness against side-channel attacks.

Acknowledgment

We are thankful to Professor Christiane Frougny for providing us references [21], [22]. We also thank the anonymous reviewers for their valuable comments. This work has been supported in part by a PhD grant from *Région Bretagne* and *Conseil Général des Côtes d'Armor* (ROBUSTA project) and by the PAVOIS project (ANR 12 BS02 002 01).

References

[1] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[2] H. Cohen and G. Frey, Eds., *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.

[3] V. Dimitrov and T. Cooklev, "Hybrid algorithm for the computation of the matrix polynomial $I + A + \ldots + A^{N-1}$," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 7, pp. 377–380, Jul. 1995.

[4] V. Dimitrov, G. Jullien, and W. Miller, "An algorithm for modular exponentiation," *Information Processing Letters*, vol. 66, no. 3, pp. 155–159, May 1998.

[5] ——, "Theory and applications of the double-base number system," *IEEE Trans. on Computers*, vol. 48, no. 10, pp. 1098–1106, Oct. 1999.

[6] V. Dimitrov, L. Imbert, and P. K. Mishra, "Efficient and secure elliptic curve point multiplication using double-base chains," in *Proc. 11th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, ser. LNCS, vol. 3788. Springer, Dec. 2005, pp. 59–78.

curves	references	curve-level operations						
		ADD	mADD	DBL	TPL	QPL	SPL	EPL
$a \neq -3$	EFD	$\mathbf{11M + 5S}$	$\mathbf{7M + 4S}$	$\mathbf{1M + 8S}$	$\mathbf{5M + 10S}$	n. a.	n. a.	n. a.
	[30]	n. a.	n. a.	$\mathbf{1M + 8S}$	$\mathbf{5M + 10S}$	$\mathbf{7M + 16S}$	$15M + 24S$	$\mathbf{17M + 30S}$
	[31]	$\mathbf{11M + 5S}$	$\mathbf{7M + 4S}$	$2M + 8S$	$6M + 11S$	$9M + 15S$	$\mathbf{13M + 18S}$	n. a.
$a = -3$	EFD	$\mathbf{11M + 5S}$	$\mathbf{7M + 4S}$	$\mathbf{3M + 5S}$	$\mathbf{7M + 7S}$	n. a.	n. a.	n. a.
	[32]	$\mathbf{11M + 5S}$	$\mathbf{7M + 4S}$	$\mathbf{3M + 5S}$	$\mathbf{7M + 7S}$	$11M + 11S$	$18M + 11S$	$\mathbf{28M + 15S}$
	[16], [31]	$\mathbf{11M + 5S}$	$\mathbf{7M + 4S}$	$\mathbf{3M + 5S}$	$7M + 8S$	$\mathbf{10M + 12S}$	$\mathbf{14M + 15S}$	n. a.

curves	references	λDBL	λTPL
$a \neq -3$	[6], [11], [33]	$4\lambda M + (4\lambda + 2)S$	$(11\lambda - 1)M + (4\lambda + 2)S$

curves	references	λTPL / λ'DBL
$a \neq -3$	[6], [11]	$(11\lambda + 4\lambda' - 1)M + (4\lambda + 4\lambda' + 3)S$

TABLE IX
COSTS OF CURVE-LEVEL OPERATIONS FROM LITERATURE AND CURVES OVER $\mathbb{F}_p$

pre-computations	methods	computation time		
			$a \neq -3$	$a = -3$
$3P$	3NAF	1DBL + 1mADD	49.4M	49.0M
	DBNS	1TPL	44.8M	44.4M
$3P$, $5P$, $7P$	4NAF	2DBL + 3mADD	140.8M	140.0M
	DBNS	1TPL + 1DBL + 2mADD	136.2M	135.4M

TABLE X
TYPICAL COSTS OF PRE-COMPUTATIONS FOR ADDITIONAL POINTS

references	methods	performances	pre-computations		recoding
			storage	operations	
	double-and-add	1 985.3M	$\emptyset$	$\emptyset$	$\emptyset$
	NAF	1 723.0M	$\emptyset$	$\emptyset$	on-the-fly & very cheap
	3NAF	1 583.7M	1 point	49.4M	on-the-fly & very cheap
	4NAF	1 499.1M	3 points	140.8M	on-the-fly & very cheap
[6]	DBNS	1 863.0M	$\emptyset$	$\emptyset$	off-line & costly
[11]	DBNS	1 722.3M	$\emptyset$	$\emptyset$	off-line & costly
[9]	DBNS	1 558.4M	7 points	>150M	off-line & costly
[7]	DBNS	1 615.3M	$\emptyset$	$\emptyset$	off-line & costly
this work	$(2, 3)$ MBNS	1 746.2M	$\emptyset$	$\emptyset$	on-the-fly & small
	$(2, 3, 5)$ MBNS	1 679.9M	$\emptyset$	$\emptyset$	on-the-fly & small
	$(2, 3, 5, 7)$ MBNS	1 670.4M	$\emptyset$	$\emptyset$	on-the-fly & small

TABLE XI
COMPARISON OF SCALAR MULTIPLICATION METHODS (CURVES OVER $\mathbb{F}_p$ WITH $a \neq -3$ AND $n = 160$)

[7] C. Doche and L. Imbert, "Extended double-base number system with applications to elliptic curve cryptography," in *Proc. 7th International Conference on Cryptology (INDOCRYPT)*, ser. LNCS, vol. 4329. Kolkata, India: Springer, Dec. 2006, pp. 335–348.

[8] R. Barua, S. K. Pandey, and R. Pankaj, "Efficient window-based scalar multiplication on elliptic curves using double-base number system," in *Proc. 8th International Conference on Progress in Cryptology (INDOCRYPT)*, ser. LNCS, vol. 4859. Springer, Dec. 2007, pp. 351–360.

[9] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters, "Optimizing double-base elliptic-curve single-scalar multiplication," in *Proc. 8th International Conference on Progress in Cryptology (INDOCRYPT)*, ser. LNCS, vol. 4859. Springer, Dec. 2007, pp. 167–182.

[10] C. Doche and L. Habsieger, "A tree-based approach for computing double-base chains," in *Proc. 13th Australasian Conference on Information Security and Privacy*, ser. LNCS, vol. 5107, Jul. 2008, pp. 433–446.

[11] V. Dimitrov, L. Imbert, and P. K. Mishra, "The double-base number system and its application to elliptic curve cryptography," *Mathematics of Computation*, vol. 77, no. 262, pp. 1075–1104, Apr. 2008.

[12] S. Singha, A. Ghosh, and A. Sinha, "A new architecture for FPGA based implementation of conversion of binary to double base number system (DBNS) using parallel search technique," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 5, pp. 12–18, Dec. 2011.

[13] T. Chabrier, D. Pamula, and A. Tisserand, "Hardware implementation of DBNS recoding for ECC processor," in *Asilomar Conference on Signals, Systems and Computers*. IEEE, Nov. 2010, pp. 1129–1133.

[14] P. K. Mishra and V. Dimitrov, "Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation," in *Proc. 10th International Conference on Information*

references	methods	performances	pre-computations		recoding
			storage	operations	
	double-and-add	1 922.0M	$\emptyset$	$\emptyset$	$\emptyset$
	NAF	1 659.7M	$\emptyset$	$\emptyset$	on-the-fly & very cheap
	3NAF	1 520.2M	1 point	49.0M	on-the-fly & very cheap
	4NAF	1 436.1M	3 points	140.0M	on-the-fly & very cheap
[7]	DBNS	1 563.2M	$\emptyset$	$\emptyset$	off-line & costly
[9]	DBNS	1 504.3M	7 points	>150M	off-line & costly
[14]	$(2,3,5)$MBNS	1 645.4M	$\emptyset$	$\emptyset$	off-line & costly
		1 606.4M	1 point	≈45M	off-line & costly
		1 566.4M	3 points	≈150M	off-line & costly
		1 552.3M	7 points	>150M	off-line & costly
		1 486.4M	5 points	>150M	off-line & costly
[32]	$(2,3)$NAF	1 514.0M	$\emptyset$	$\emptyset$	small
	$(2,3,5)$NAF	1 490.0M	$\emptyset$	$\emptyset$	small
	$(2,3,5,7)$NAF	1 491.0M	$\emptyset$	$\emptyset$	small
	$(2,3)$NAF$_3$	1 460.0M	1 point	≈45M	small
	$(2,3,5)$NAF$_3$	1 444.0M	1 point	≈45M	small
	$(2,3,5,7)$NAF$_3$	1 449.0M	1 point	≈45M	small
	$(2,3)$NAF$_4$	1 384.0M	3 points	>150M	small
	$(2,3,5)$NAF$_4$	1 383.0M	3 points	>150M	small
	$(2,3,5,7)$NAF$_4$	1 394.0M	3 points	>150M	small
[16]	$(2,3,5)$NAF	1 460.0M	$\emptyset$	$\emptyset$	costly
	$(2,3,5)$NAF	1 426.0M	6 points	>150M	costly
this work	$(2,3)$MBNS	1 686.2M	$\emptyset$	$\emptyset$	on-the-fly & small
	$(2,3,5)$MBNS	1 631.0M	$\emptyset$	$\emptyset$	on-the-fly & small
	$(2,3,5,7)$MBNS	1 629.3M	$\emptyset$	$\emptyset$	on-the-fly & small

TABLE XII
COMPARISON OF SCALAR MULTIPLICATION METHODS (CURVES OVER $\mathbb{F}_p$ WITH $a = -3$ AND $n = 160$)

Security (ISC), ser. LNCS, vol. 4779, Oct. 2007, pp. 390–406.

[15] P. Longa, "Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields," Master's thesis, Univ. Ottawa, 2007.

[16] P. Longa and C. Gebotys, "Fast multibase methods and other several optimizations for elliptic curve scalar multiplication," in *Proc. Public Key Cryptography (PKC)*, ser. LNCS, vol. 5443, 2009, pp. 443–462.

[17] G. N. Purohit and A. S. Rawat, "Fast scalar multiplication in ECC using the multi base number system," *International Journal of Computer Science Issues*, vol. 8, no. 1, pp. 131–137, May 2011.

[18] X. Yin, T. Yang, and J. Ning, "Optimized approach for computing multi-base chains," in *Proc. 7th International Conference on Computational Intelligence and Security (CIS)*. IEEE, Dec. 2011, pp. 964–968.

[19] J. Adikari, V. S. Dimitrov, and L. Imbert, "Hybrid binary-ternary number system for elliptic curve cryptosystems," *IEEE Transactions on Computers*, vol. 60, no. 2, pp. 254–265, Feb. 2011.

[20] G. Purohit, A. S. Rawat, and M. Kumar, "Elliptic curve point multiplication using MBNR and point halving," *International Journal of Advanced Networking and Applications*, vol. 3, no. 5, pp. 1329–1337, 2012.

[21] B. Pascal, *Œuvres complètes*. Librarie Lefèvre, 1819, vol. 5, ch. De Numeribus Multiplicibus, pp. 117–128.

[22] J. Sakarovitch, *Elements of Automata Theory*. Cambridge, 2009, ch. Prologue: M. Pascal's Division Machine, pp. 1–6.

[23] T. Jebelean, "An algorithm for exact division," *Journal of Symbolic Computation*, vol. 15, no. 2, pp. 169–180, Feb. 1993.

[24] N. Boullis and A. Tisserand, "Some optimizations of hardware multiplication by constant matrices," *IEEE Transactions on Computers*, vol. 54, no. 10, pp. 1271–1282, Oct. 2005.

[25] A. Byrne, E. Popovici, and W. Marnane, "Versatile processor for gf(p^m) arithmetic for use in cryptographic applications," *IET Computers & Digital Techniques*, vol. 2, no. 4, pp. 253–264, Jul. 2008.

[26] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[27] E. Oswald, *Advances in Elliptic Curve Cryptography*, ser. London Mathematical Society Lecture Note Series. Cambridge University Press, Apr. 2005, vol. 317, ch. Side Channel Analysis, pp. 69–86.

[28] M. Joye, *Advances in Elliptic Curve Cryptography*, ser. London Mathematical Society Lecture Note. Cambridge University Press, Apr. 2005, vol. 317, ch. Defenses Against Side-Channel Analysis, pp. 87–100.

[29] A. Byrne, N. Meloni, A. Tisserand, E. M. Popovici, and W. P. Marnane, "Comparison of simple power analysis attack resistant algorithms for an elliptic curve cryptosystem," *Journal of Computers*, vol. 2, no. 10, pp. 52–62, 2007.

[30] P. Giorgi, L. Imbert, and T. Izard, "Optimizing elliptic curve scalar multiplication for small scalars," in *Proc. Mathematics for Signal and Information Processing*, vol. 7444. SPIE, Sep. 2009, pp. 74 440N:1–10.

[31] P. Longa and C. Gebotys, "Setting speed records with the (fractional) multibase non-adjacent form method for efficient elliptic curve scalar multiplication," Cryptology ePrint Archive, Tech. Rep. 118, 2008.

[32] P. Longa and A. Miri, "New multibase non-adjacent form scalar multiplication and its application to elliptic curve cryptosystems," IACR Eprint, Tech. Rep. 52, 2008.

[33] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 1717. Springer, Aug. 1999, pp. 61–72.