Short Memory Scalar Multiplication on Koblitz Curves

Katsuyuki Okeya[1], Tsuyoshi Takagi[2], and Camille Vuillaume[1]

[1] Hitachi, Ltd., Systems Development Laboratory, Kawasaki, Japan.
{ka-okeya,camille}@sdl.hitachi.co.jp
[2] Future University - Hakodate, Japan.
takagi@fun.ac.jp

Abstract. We present a new method for computing the scalar multiplication on Koblitz curves. Our method is as fast as the fastest known technique but requires much less memory. We propose two settings for our method. In the first setting, well-suited for hardware implementations, memory requirements are reduced by 85%. In the second setting, well-suited for software implementations, our technique reduces the memory consumption by 70%. Thus, with much smaller memory usage, the proposed method yields the same efficiency as the fastest scalar multiplication schemes on Koblitz curves.

Keywords: *elliptic curve cryptosystems, Koblitz curves, scalar multiplication, NAF, polynomial basis, normal basis, change-of-basis, smartcard.*

1 Introduction

Elliptic curves cryptosystems (ECC) offer an interesting alternative to standard prime-field based cryptosystem, because for the same security level, they are much faster and require less memory [11, 13]. In particular, they are well-suited for implementations on low-end processors and memory-constrained devices such as smartcards. From the geometrical properties of elliptic curves, one can define two operations on the points of the curve: point addition and point doubling. Then, given a base point P and a scalar d, one can compute the scalar multiplication $Q = dP$. It is believed that the discrete logarithm problem on elliptic curves (EC-DLP), namely finding d from P and Q, is a hard problem, and many cryptosystems rely on the hardness of the EC-DLP. Since the scalar multiplication plays a crucial role in such cryptosystems, it is important to implement it efficiently. In practical cases, we distinguish two types of scalar multiplications. On the one hand, the scalar multiplication with known base point can be computed very efficiently. On the other hand, when the base point is random, much more computational effort is required.

A first approach to decrease the computational cost of the scalar multiplication with unknown base point is to deploy a new representation of the scalar in order to minimize the number of elliptic operations. The standard technique is to precompute some small multiples of the base point P and re-use these points

in the scalar multiplication. One of the fastest recoding technique is the width w non-adjacent form (NAF_w), which reduces the number of point additions to $m/(w+1)$ on average for a m-bit scalar, provided that 2^{w-2} points are precomputed. Unfortunately, recoding techniques have no influence on the number of point doublings, and as a consequence, the speed-up which arises from the NAF_w is limited.

A second approach is to use special curves. In particular, on some special binary curves called Koblitz curves, all point doublings can be replaced by a much cheaper operation: the Frobenius automorphism [12]. Thus, NAF_w techniques and Koblitz curves nicely combine: the NAF_w reduces the number of point additions while point doublings are eliminated thanks to special algebraic properties of the curve [18]. Since the computational cost of precomputations grows with 2^{w-2} whereas that of the scalar multiplication itself decreases with $1/(w+1)$, obviously, there is an optimal value for w. In practical situations, $w = 5$ is optimal for Koblitz curves.

In [1], a method for computing the scalar multiplication on Koblitz curves without precomputations was proposed. The technique is faster than the NAF_2 on Koblitz curves, but slower than the NAF_3 which has one precomputed point. Additionally, it is well-known that on binary field, with special bases called normal bases, squares are virtually free. Techniques based on normal basis representations have been proposed to speed-up the scalar multiplication on Koblitz curves, with known point and large memory [4].

We propose a new method which can exploit the speed-up arisen from precomputations without actually storing all of the precomputed points: we keep the full power of precomputations without sacrifice on the side of memory. In fact, we embed the precomputation process in the scalar multiplication: the computations of the scalar multiplication are re-ordered and the precomputed points are generated sequentially. In our method, the order of computations does not play any role in the global efficiency, and operations can be freely re-ordered. Therefore, our scheme can be performed from right to left, or from left to right, or even following a random sequence, generalizing the notion of right-to-left or left-to-right computations. Our algorithm has two different settings. With the first setting, which is well-suited for hardware implementations, our method is as fast as the (optimal) NAF_5 but requires only one auxiliary point instead of seven, reducing the memory consumption by 85%. The second setting is well-suited for software implementations, that is, for general-purpose processors, but requires two auxiliary points instead of seven, reducing memory consumption by 70%.

2 Preliminaries

In this section, we discuss known facts: we describe the properties of polynomial and normal basis implementations of binary fields, and introduce Koblitz curves.

2.1 Polynomial Bases vs. Normal Bases

In the field $\mathbb{F}_{2^m}$, elements are represented with respect to a basis $(\epsilon_0, \ldots, \epsilon_{m-1})$, where $\epsilon_i \in \mathbb{F}_{2^m}$. Then, the element $b \in \mathbb{F}_{2^m}$ is represented with the vector with binary entries $(b_0 \ldots b_{m-1})_2$, corresponding to the polynomial $\sum_{i=0}^{m-1} b_i \epsilon_i$. For efficient implementations, two types of bases are usually considered: polynomial and normal bases.

In a polynomial basis, ϵ_i is the monomial X^i of degree i, and operations are computed modulo an irreducible polynomial $\Pi[X]$ of degree m, appropriately chosen in order to speed-up reduction. Also, there are efficient algorithms for multiplications and squarings in $\mathbb{F}_{2^m}$ using polynomial bases [6]. A normal basis consists of the m-tuple $(\beta_0, \ldots, \beta_{m-1})$, where $\beta_i^2 = \beta_{i+1}$ for $i < m-1$ and $\beta_{m-1}^2 = \beta_0$. In other words, powers of two are simple cyclic shifts. Unfortunately, normal basis multiplications are slow. On the one hand, in hardware, with an appropriate circuitry and good parameter choices, the penalty arisen from slow multiplications can be minimized and normal bases yield an elegant and efficient solution to implement binary fields. On the other hand, in software, polynomial bases largely outperform normal bases [5].

Even though a full normal basis approach seems too slow to compete with polynomial basis implementations, with a mixed normal-polynomial approach, one can benefit from the advantages of both types of bases: fast multiplications with polynomial bases and fast computation of powers of two with normal bases. Indeed, there are techniques to convert elements between their normal and polynomial basis representations, using change-of-basis matrices [7]. The conversion time is roughly the same as one polynomial basis multiplication, and each matrix occupies m^2 bits in memory [4]. Note that change-of-basis matrices can be precomputed off-line and stored in ROM.

2.2 Koblitz Curves

Koblitz curves belong to a special class of elliptic curves defined over binary fields, and additionally offer a very efficient arithmetic, with no significant security flaw compared to general binary curves [12]. They are defined over a binary field $\mathbb{F}_{2^m}$ by the equation:

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \tag{1}$$

where $a \in \{0, 1\}$. We denote by $E_a(\mathbb{F}_{2^m})$ the abelian group of the points of the Koblitz curve over $\mathbb{F}_{2^m}$, along with the point of infinity $\mathcal{O}$, neutral element of the addition law.

The main interest of Koblitz curves is that point doublings can be totally eliminated from the scalar multiplication, and replaced by the efficiently computable Frobenius automorphism $\Phi : (x, y) \mapsto (x^2, y^2)$. Since the quadratic equation $(x^4, y^4) + 2(x, y) = \mu(x^2, y^2)$ where $\mu = (-1)^{1-a}$ holds for every point of Koblitz curves, the Frobenius map can be seen as the complex $\tau = (\mu + \sqrt{-7})/2$, solution of the equation $\Phi^2 + 2 = \mu\Phi$. The approach for fast computations

over Koblitz curves is to convert a scalar d to a radix τ expansion such as $d = \sum_{i=0}^{j} d_i \tau^i,\ d_i \in \{0, \pm 1\}$: in the scalar multiplication, point doublings are replaced by the efficiently computable Frobenius map. However, in order to fully take advantage of the Frobenius map, the τ expansion must be sparse and short. In [18], Solinas proposed two efficient algorithms to satisfy these properties: partial reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$ and radix-τ NAF recoding.

One can see radix-τ integers as elements of the quadratic field $\mathbb{Z}[\tau]$, with norm $|\lambda| = l_0^2 + \mu l_0 l_1 + 2l_1^2$. Then the division with remainder $\gamma = \kappa \cdot \delta + \rho$ in $\mathbb{Z}[\tau]$ satisfies $|\rho| \leq 4/7 * |\delta|$. Here $\zeta P = (\zeta \bmod \delta)P$ holds for $\delta = (\tau^m - 1)/(\tau - 1)$ and $\zeta \in \mathbb{Z}[\tau]$, and thus the radix τ expansion of reduced scalars becomes shorter.

To generate a width w radix τ NAF expansion (TNAF_w), one can use the map $\Phi_w : u_0 + u_1 \cdot \tau \in \mathbb{Z}[\tau] \mapsto u_0 + u_1 \cdot t_w \bmod 2^w \in \mathbb{Z}/2^w\mathbb{Z}$, where $t_w = 2U_{w-1}U_w^{-1} \bmod 2^w$, U_w is the Lucas sequence U_w, defined by $U_0 = 0$, $U_1 = 1$ and $U_{w+1} = \mu U_w - 2U_{w-1}$ for $w \geq 1$, and "mods 2^w" means the signed residue modulo 2^w. Interestingly, the odd representatives modulo τ^w correspond exactly to the odd integers in $\mathbb{Z}/2^w\mathbb{Z}$ by Φ_w [18]. Therefore, the congruence classes modulo τ^w can be easily computed using Φ_w.

Algorithm 1: TNAF_w with partial modular reduction modulo δ [18]

INPUT: Scalar d ;
OUTPUT: TNAF_w(s);

1. $c_0 \leftarrow d$ partmod δ (partial modular reduction); $c_1 \leftarrow 0$; $i \leftarrow 0$;
2. **while** $c_0 \neq 0$ **or** $c_1 \neq 0$ **do**
   (a) **if** c_0 is odd **then** $u \leftarrow \Phi_w(c_0 + c_1\tau)$ **else** $u \leftarrow 0$;
   (b) $d_i^{(w)} \leftarrow u$; $c_0 \leftarrow c_0 - u$;
   (c) $(c_0, c_1) \leftarrow (c_1 + \mu c_0/2, -c_0/2)$; $i \leftarrow i + 1$;
3. **return** $(d_{i-1}^{(w)} \ldots d_1^{(w)} d_0^{(w)})$;

If the scalar is first reduced modulo δ, the length of the TNAF_w is at most $m+a+3$, and does not exceed $m+a$ with high probability [18]. Since the average nonzero density of the TNAF_w is $1/(w+1)$, the scalar multiplication requires $(m+a)/(w+1)$ point additions. Additionally, the points $P, 3P, \ldots, (2^{w-1} - 1)P$ must be precomputed. Therefore, the total cost of the scalar multiplication using TNAF_w is on average:

$$\mathcal{C}_{NAF} = (m + a) \cdot \mathrm{ECFRB} + \left(\frac{m + a}{w + 1} + 2^{w-2} - 1 \right) \cdot \mathrm{ECADD} + \mathrm{ECDBL} \quad (2)$$

where ECADD, ECFRB and ECDBL stand for the computational cost of point additions, τ multiplications and point doublings, respectively.

3 Short Memory Scalar Multiplication

We present the basic idea to compute the TNAF_5 with memory for only one auxiliary point instead of the seven usual precomputed points, assuming that a normal basis is used for representing elements of the underlying field.

3.1 Motivation

The general approach to decrease the cost of the scalar multiplication for an unknown base point is to precompute some multiples of the base point in order to minimize the number of point additions in the scalar multiplication. This method is particularly effective on Koblitz curves where point additions *only* are significant for the total computational cost. Especially, in the case of the TNAF_w method, for several binary fields of cryptographic interest ($m = 163, 233, 283, 409$), the width $w = 5$ is optimal[3]. Unfortunately, in practical situations, scarce memory resources might prevent us from choosing the optimal width. For example, for $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$ and $\mathbb{F}_{2^{409}}$, the storage of 7 precomputed points occupies 294, 420, 504 and 728 bytes in RAM, respectively. Newer smartcards have larger RAM space, but the current trend is to develop multi-application cards with greedy memory requirements on the OS side. Generally, independently from how much RAM is available, only about 500 bytes are allocated for cryptography in total. Thus, having an important part, or even worse, the totality of the allocated memory occupied by precomputed values can be a serious drawback. Besides, in hardware and in software, it is advantageous to decrease memory requirements, and as a consequence, decrease the cost of the final device.

We propose a solution to drastically reduce the memory requirements of the optimal TNAF_5 scalar multiplication. Our approach is as follows: we re-group calculations involving the same precomputed point, which can be discarded immediately after such calculations are completed.

3.2 Sequential precomputations

Since we aim at discarding precomputed points when they are not needed anymore, we need a procedure for computing them sequentially.

Definition 1. *We call sequential precomputations an ordered sequence of points where the kth point P_k can be computed from the base point P and the previous point P_{k-1}, with one point addition and several τ multiplications.*

In [18], the precomputed points $3P, 5P, \ldots, (2^{w-1} - 1)P$ are calculated by successively adding $2P$. Since the non-trivial point $2P$ must be readily available, the precomputations are not sequential in the sense of Definition 1. The computational cost of this technique is $(2^{w-2} - 1)\mathrm{ECADD} + \mathrm{ECDBL}$. Instead

[3] On general curves, one can achieve a greater speed-up by using fractional width techniques. However, such a method has not been proposed yet for Koblitz curves.

of $u = \pm 1, \ldots, \pm(2^{w-1} - 1)$, one can take $(u \bmod \tau^w)$ as coefficients in the TNAF_w. The precomputed points $(u \bmod \tau^w)P$ can be efficiently calculated thanks to relationships between the TNAF_2 representations of $(u \bmod \tau^w)$, for $u = 1, \ldots, (2^{w-1} - 1)$. Then, the computational cost of precomputations can be reduced to only $2^{w-2} - 1$ point additions [6]. With this technique, each point $(u \bmod \tau^w)P$ can be computed independently for $w < 5$: sequential precomputations are trivially possible for the TNAF_2, TNAF_3 and TNAF_4. However, for $w = 5$, it is necessary to store $(5 \bmod \tau^5)P$ during the whole precomputation process, which is unacceptable for our aims: we do not wish to store any non-trivial point. This is unfortunate because $w = 5$ is usually the optimal width. Thus, we have to look for new a technique which allows us to compute the points sequentially in the TNAF_5.

Unlike the standard method which utilizes the TNAF_2 representations of $(u \bmod \tau^5)$ for the precomputations, we use the binary expansion of $u \bmod \tau^5$. Then, we propose a precomputation technique which requires $2^{w-2} - 1$ point additions and several τ multiplications, with memory for only one non-trivial point.

Table 1. Odd representatives modulo τ^5

	Case $a = 0$		**Case** $a = 1$	
u	$\alpha_u = u \bmod \tau^5$	binary rep. of α_u	$\alpha_u = u \bmod \tau^5$	binary rep. of α_u
1	1	1	1	1
3	$-\tau - 3$	$\tau^2 - 1$	$\tau - 3$	$\tau^2 - 1$
5	$-\tau - 1$	$-\tau - 1$	$\tau - 1$	$\tau - 1$
7	$-\tau + 1$	$-\tau + 1$	$\tau + 1$	$\tau + 1$
9	$-2\tau - 3$	$-\tau^4 + \tau - 1$	$2\tau - 3$	$-\tau^4 - \tau - 1$
11	$-2\tau - 1$	$\tau^3 + \tau^2 - 1$	$2\tau - 1$	$-\tau^3 + \tau^2 - 1$
13	$-2\tau + 1$	$\tau^3 + \tau^2 + 1$	$2\tau + 1$	$-\tau^3 + \tau^2 + 1$
15	$3\tau + 1$	$-\tau^3 - \tau^2 + \tau + 1$	$-3\tau + 1$	$\tau^3 - \tau^2 - \tau + 1$

Table 1 shows the representatives modulo τ^5 and their binary expansion, for the cases $a = 0$ and $a = 1$. Interestingly, one can find sequential relationships between the representatives: for instance, $\alpha_{11} = \tau^3 + \alpha_3$ and $\alpha_{15} = \tau - \alpha_{11}$. Indeed, thanks to these relationships, it is possible to compute the points $(u \bmod \tau^5)P$ sequentially: Table 2 presents a practical solution for such sequential precomputations, following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$. Then, the computational cost of our precomputation technique using a normal and a polynomial basis is respectively:

$$\mathcal{C}_{0,n}^{(5)} = 7 \cdot \mathrm{ECADD}_n + 7 \cdot \mathrm{ECFRB}_n \text{ and } \mathcal{C}_{0,p}^{(5)} = 7 \cdot \mathrm{ECADD}_p + 14 \cdot \mathrm{ECFRB}_p, \quad (3)$$

where the indexes n and p stand for the type of basis (normal or polynomial). In the following, the index 0 in $\mathcal{C}_0$ will refer to the cost of precomputations.

Table 2. Sequential computation of $(u \bmod \tau^5)P$

u	$\alpha_u P = (u \bmod \tau^5)P, u = 0$	$\alpha_u P = (u \bmod \tau^5)P, u = 1$
1	$F(1, P, -^{\text{a}}) = \alpha_1 P = P$	$F(1, P, -) = \alpha_1 P = P$
3	$F(3, P, -) = \alpha_3 P = \tau^2 P - P$	$F(3, P, -) = \alpha_3 P = \tau^2 P - P$
11	$F(11, P, \alpha_3 P) = \alpha_{11} P = \tau^3 P + \alpha_3 P$	$F(11, P, \alpha_3 P) = \alpha_{11} P = -\tau^3 P + \alpha_3 P$
15	$F(15, P, \alpha_{11} P) = \alpha_{15} P = \tau P - \alpha_{11} P$	$F(15, P, \alpha_{11} P) = \alpha_{15} P = -\tau P - \alpha_{11} P$
5	$F(5, P, -) = \alpha_5 P = -\tau P - P$	$F(5, P, -) = \alpha_5 P = \tau P - P$
13	$F(13, P, \alpha_5 P) = \alpha_{13} P = P - \tau^2 \alpha_5 P$	$F(13, P, \alpha_5 P) = \alpha_{13} P = P - \tau^2 \alpha_5 P$
7	$F(7, P, -) = \alpha_7 P = -\tau P + P$	$F(7, P, -) = \alpha_7 P = \tau P + P$
9	$F(9, P, \alpha_7 P) = \alpha_9 P = -\tau^4 P - \alpha_7 P$	$F(9, P, \alpha_7 P) = \alpha_9 P = -\tau^4 P - \alpha_7 P$

[a] The symbol "$-$" refers to any point, as the third input parameter is not used in this case.

3.3 Short Memory Scalar Multiplication with a Normal Basis

On standard elliptic curves, the precomputation work must be done before the scalar multiplication itself; all precomputed points are stored in RAM. On Koblitz curves implemented with a normal basis, we will show that this is unnecessary. Indeed, when P is known, $\tau^i P$ can be computed with i-fold cyclic shifts of the coordinates of P. In other words, the order of the computations of the scalar multiplication $dP = \sum_{i=0}^{m+a} d_i \tau^i P$ does not matter. Especially, one can re-group calculations involving the same precomputed points, as shown in Algorithm 2.

The average computational cost of Algorithm 2 is:

$$\mathcal{C}_{1,n}^{(5)} = \frac{m+a}{6} \cdot \text{ECADD}_n + \left(\frac{m+a}{6} + 4 \right) \cdot \text{ECFRB}_n + \mathcal{C}_{0,n}^{(5)}. \tag{4}$$

Where the index 1 in $\mathcal{C}_1$ stands for the total computational cost, whereas 0 in $\mathcal{C}_0$ stands for the cost of precomputations. It is exactly the same as that of the standard TNAF_5, with 4 additional τ-multiplications arising from step 2(c), which re-sets the current precomputed point to its original value $\alpha_u P$ in order to evaluate the next precomputed point. Interestingly, Algorithm 2 computes the scalar multiplication using the (optimal) TNAF_5 with only one auxiliary point, namely the current precomputed point R. In comparison, the standard TNAF_5 requires 7 non-trivial (that is, different from P) precomputed points. Thus, for a completely negligible overhead (4 ECFRB), we reduced the memory

Algorithm 2: Short memory scalar multiplication on a normal basis

INPUT: Base point P, scalar d;
OUTPUT: $Q = dP$;

1. compute $d^{(5)} = \text{TNAF}_5(d)$; $Q \leftarrow \mathcal{O}$; $R \leftarrow \mathcal{O}$;
2. **for** u following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$ **do**
   (a) $R \leftarrow F(u, P, R)$ with Table 2; $k \leftarrow 0$;
   (b) **for** j **from** 0 **to** $m + a - 1$ **do**
      i. **if** $|d_j^{(5)}| = u$ **then**
         A. $R \leftarrow \tau^{j-k} R$; $k \leftarrow j$;
         B. $Q \leftarrow Q + \text{sign}(d_j^{(5)})R$;
   (c) **if** $u \in \{3, 11, 5, 7\}$ **then** $R \leftarrow \tau^{m-k} R$;
3. **return** Q;

consumption[4] by a factor 7. Or, putting it in a different way, for the same memory consumption as that of the TNAF_3, our method is as fast as the TNAF_5. Note that since sequential precomputations are trivially possible for $w = 2, 3, 4$, our technique is also applicable to the TNAF, TNAF_2, TNAF_3 and TNAF_4.

4 Short Memory with Change-of-Basis

In the following we show how a mixed normal-polynomial basis approach allows us to deploy the short memory method not only on hardware implementations but also on general-purpose processors.

4.1 General Idea

The interest of the latter idea is practically limited by the fact that a normal basis is necessary in order to efficiently compute multiplications by τ^i. In software, normal basis implementations are much slower than polynomial basis implementations. At this point, it would be interesting to take the best from the two approaches: fast computations of Frobenius map with a normal basis and fast field multiplications with a polynomial basis, and convert between the two representations when necessary. This mixed approach was already used for implementing a fast generator of pairs (k, kP) for signature schemes [4], and for defeating side channel attacks [16], but never to speed-up the scalar multiplication itself.

The general idea of our technique is to perform sequential precomputations with a polynomial basis, but then, convert the auxiliary precomputed point R_p to its normal basis representation R_n. After that, $\tau^i R_n$ can be computed with

[4] By memory consumption, we refer to non-trivial precomputed points: we do not consider buffers, nor the base point or accumulators.

only two cyclic shifts, for any i. The point R_n with shifted coordinates can be converted back to its polynomial basis representation in order to perform point additions with the polynomial basis.

We remark an interesting property of this approach: to convert a point R_n represented with respect to the normal basis to its polynomial basis representation, one only needs the binary expansion of the coordinates of $R_n = (x_n, y_n)$. More precisely, the conversion to the polynomial basis is as follows: if the kth bit of x_n is 1, then add (i.e. xor) the kth line of the change-of-basis matrix to x_p. Of course, the same holds for y_n and y_p. As a consequence, it is not needed to explicitly compute $\tau^i R_n$: the knowledge original (unshifted) binary expansion of x_n and y_n is sufficient to convert $\tau^i R_n$ to its polynomial basis representation. Indeed, if the kth bit of x_n is 1, we simply add the $(k + i \bmod m)$ line of the change-of-basis matrix to x_p, and the same holds for y_n and y_p.

4.2 Short Memory TNAF$_5$ with Change-of-Basis

Unlike the normal basis short memory method, we use two auxiliary points instead of just one, in order to avoid additional conversions to the normal basis. More precisely, the auxiliary precomputed point is stored in its polynomial and normal basis representations. Then, the average computational cost of the scalar multiplication using Algorithm 3 is:

$$\mathcal{C}^{(5)}_{1,cob} = \frac{m+a}{6} \cdot \mathrm{ECADD}_p + \left(\frac{m+a}{6} + 12\right) \cdot \mathrm{COB} + \mathcal{C}^{(5)}_{0,p}, \tag{5}$$

where COB stands for the computational cost for changing the basis. Note that the auxiliary precomputed point R_p is represented in affine coordinates, and thus, it is possible to use mixed projective-affine additions formulas [6].

Algorithm 3: Short memory scalar multiplication with change of basis, $w = 5$

INPUT: Base point P, scalar d;
OUTPUT: $Q = dP$;

1. compute $d^{(5)} = \mathrm{TNAF}_5(d)$; $Q \leftarrow \mathcal{O}$;
2. **for** u following the sequence $\{1, 3, 11, 15, 5, 13, 7, 9\}$ **do**
   (a) $R_p \leftarrow (u \bmod \tau^5)P$ with affine coordinates, polynomial basis;
   (b) $R_n \leftarrow$ convert R_p to normal basis;
   (c) **for** j **from** 0 **to** $m + a - 1$ **do**
      i. **if** $|d^{(5)}_j| = u \bmod \tau^5$ **then**
         A. $R_p \leftarrow$ convert $\tau^j R_n$ to polynomial basis;
         B. $Q \leftarrow Q + \mathrm{sign}(d^{(5)}_j) R_p$ with polynomial basis, mixed coordinates;
   (d) **if** $u \in \{3, 11, 5, 7\}$ **then** $R_p \leftarrow$ convert R_n to polynomial basis;
3. **return** Q;

On the one hand, in the original TNAF_5 scalar multiplication computed with a polynomial basis, the Frobenius map must be explicitly computed, requiring $3 \cdot (m + a)$ squarings assuming projective coordinates. On the other hand, the short memory method does not compute the Frobenius map explicitly, but instead, several conversions to/from the polynomial basis and the normal basis are needed: one each time the auxiliary precomputed point is updated (i.e. 8), one before each point addition (i.e. $(m + a)/6$) and one when the new value of the auxiliary point requires the previous auxiliary point (i.e. 4). Since squarings and change-of-basis operations have a small computational cost compared to point additions, we can expect the original TNAF_5 and our method to have similar running times. In fact, depending on the relative speed of squarings and change-of-basis operations, the short memory method might be slightly slower or faster than the TNAF_5.

5 Comparisons and Properties

Finally, we discuss side channel attacks, generalize the computation strategy described in Algorithms 2 and 3, consider the two cases of hardware and software implementations, and show how the method compares with known techniques.

5.1 Discussion on Side Channel Attacks

On low-end processors running cryptography, side channel attacks are a serious threat. By re-grouping calculations involving the same precomputed point, the short memory method is naturally weaker than other methods, leaking even more information.

However, there are many situations where side channel analysis is not an issue. Consider EC-DSA, for instance. On the one hand, the signature generation should be protected against side channel attacks, as it involves secret parameters. Additionally, the signature generation is based on one scalar multiplication with known base point, which can be computed very efficiently thanks to comb methods [15]. On the other hand, the signature verification involves computations with random points, which are much less efficient, and only publicly available parameters. This is the ideal setting for the (unprotected) short memory method. Note that the short memory method does not combine well with interleave methods, but on Koblitz curves, the benefit obtained from interleave methods is small anyway, because point doublings are already replaced by the efficiently computable Frobenius automorphism.

In some situations, the base point is random and the scalar should be kept secret. Then, one can still benefit from the advantages of the short memory method and in the same, protect the scalar multiplication against side channel attacks. There are two types of side channel attacks based on power consumption analysis: simple power analysis (SPA) and differential power analysis (DPA). In the frame of SPA, the attack rely on one single power trace, whereas several power traces are analyzed with the help of a statistical tool in the case of DPA. To thwart

SPA, one can use side channel atomicity, a technique virtually applicable to any algorithm [3]. The basic idea of side channel atomicity is to assemble atomic blocks which are indistinguishable by SPA, and implement every operation with atomic blocks. DPA can also be defeated with adequate countermeasures: the random exponent recoding technique applied to Koblitz curves [9] combines well with our method.

5.2 Recoding and Calculation Strategies

For the sake of simplicity, we introduced our technique with a repeated right-to-left scanning of the TNAF_w in Algorithms 2 and 3. However, since τ multiplications are computed with respect to a normal basis, the *order* of computations does not matter: one can compute $\tau^i P$ or $\tau^{-i} P$ with only two cyclic shifts, and the cost of the latter operation is independent from the cycle length i. In other words, instead of computing right-to-left, one can compute left-to-right, or even following a random re-ordering of the operations! Note that such alternative computation strategies have no influence on efficiency: the auxiliary point R in Algorithm 2, or R_p in Algorithm 3, is always represented with affine coordinates, whereas the accumulator Q can be represented with projective coordinates such as LD-coordinates. Therefore, one can always use mixed affine-projective formulas, independently from the computation strategy. In this sense, our technique differs from standard methods were a left-to-right computation strategy is necessary when mixed addition formulas are used.

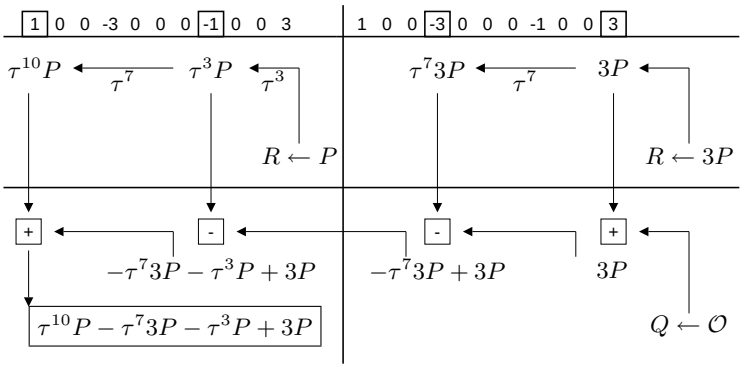

Fig. 1. Right-to-Left Computation Strategy

On-the-Fly Right-to-Left Strategy. In the standard approach, the scalar is converted to the TNAF_w using a right-to-left strategy whereas the scalar multiplication is computed left-to-right. Thus, the scalar must be stored in both of its original and TNAF_w representations, wasting $O(m)$ bits. With the short memory method, we are free from the left-to-right constraint for computations, and as a consequence, it becomes possible to recode the scalar on-the-fly, without

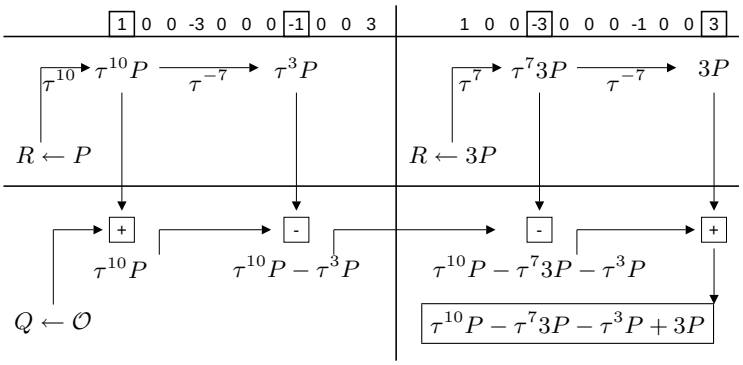

Fig. 2. Left-to-Right Computation Strategy

effectively storing the coefficients of the TNAF_w. The idea is as follows: each time the auxiliary precomputed point is updated, the recoding is performed anew. For each individual recoding, additions (or subtractions) with the auxiliary point $(u \bmod \tau^w)P$ are performed only when the corresponding coefficient of the TNAF_w is $\pm(u \bmod \tau^w)$. Of course, one has to repeat the same recoding several times, but the computational cost of the recoding process is generally negligible compared to that of the scalar multiplication itself, and in the case of the TNAF_5, the recoding must be calculated only 8 times, limiting the impact of the redundancy overhead.

Randomized Computation Sequence. For a given precomputed auxiliary point $(u \bmod \tau^w)P$, the order of computations can be freely chosen. In fact, it can even be randomized. For $w = 5$, there are some constraints on the precomputed auxiliary points themselves: $(3 \bmod \tau^5)P$ must be calculated before $(11 \bmod \tau^5)P$, for instance. But for $w < 5$, one can compute the points $(u \bmod \tau^w)P$ following any order, and then for fixed u, randomize the computation sequence again. Re-ordering operations can be used in order to prevent side-channel or fault attacks. However, this idea is based on temporal obfuscation: the order of the operations is randomized, but not the operations themselves. Therefore, alone, this technique might be insufficient to defeat side channel attacks, but combined with other countermeasures, it provides higher security for free.

5.3 Hardware Implementation

In hardware, the cylic shift is virtually free: for example, it can be emulated with a pointer on the least significant bit. Additionally, there are efficient normal basis multipliers for hardware implementations [14], and even when a polynomial basis representation is required for interoperability with other systems, the conversion from a normal basis to a polynomial basis representation can be carried out without storing the change-of-basis matrix [10]. Thus, the natural choice is a normal basis, where our method offers outstanding speed-ups with very

small memory requirements. Alternatively, by combining Frobenius expansions and the point halving method, one can obtain a non-zero density of 2/7 with no precomputations [1]. Table 3 summarizes the computational costs and memory requirements for precomputations of our technique, which has one auxiliary precomputed point, the TNAF_2 and the τ/halve method, which have no precomputed tables, and with the TNAF_3 and the TNAF_5, which have one and seven precomputed points, respectively. Clearly, the short memory method outperforms the TNAF_2, TNAF_3 and the τ/halve techniques. The TNAF_5 is as fast as the short memory, but requires seven points whereas our method needs only one auxiliary point for precomputations, reducing the memory consumption for precomputations by 85%.

Table 3. Speed and Memory Comparisons for Hardware Implementations

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
TNAF_2	54 add.	78 add.	94 add.	136 add.	190 add.
τ/halve	47 add.	67 add.	81 add.	117 add.	163 add.
TNAF_3	42 add. 326 bits	59 add. 466 bits	72 add. 566 bits	103 add. 818 bits	144 add. 1142 bits
TNAF_5	34 add. 2282 bits	46 add. 3262 bits	54 add. 3962 bits	75 add. 5726 bits	102 add. 7994 bits
Short Memory (Algorithm 2)	34 add. 326 bits	46 add. 466 bits	54 add. 566 bits	75 add. 818 bits	102 add. 1142 bits

5.4 Software Implementation

A polynomial basis implementation is usually preferred in software, because field multiplications are much faster than when using a normal basis. However, the two approaches can be combined by using a normal basis for computing the Frobenius map and a polynomial basis for computing point additions.

In Table 4, we estimate the computational cost of the short memory method with change-of-basis and compare it with standard TNAF_w techniques, assuming mixed affine-LD projective coordinates for the scalar multiplication and affine coordinates for precomputations, and that the cost of field inversions, field squarings and change-of-basis operation is equivalent to that of 10, 1/7 and 1 field multiplications, respectively. Our estimations show that under our assumptions, the short memory method is about as fast as the TNAF_5, with a small advantage for the TNAF_5 for small bitlengths. For example, for the field $\mathbb{F}_{2^{163}}$, the TNAF_5 is the fastest, but the difference with the short memory is only about 2%,

and the short memory method requires about 70% less memory. However, for greater bitlengths, the short memory method is slightly faster than the TNAF_5: the overhead introduced by the change-of-basis operations is smaller than the speed-up obtained from saving τ multiplications. Note that the situation may be different in practical implementations, depending on the relative speed of change-of-basis operations and polynomial basis squarings. In particular, an efficient change-of-basis method would definitely give the advantage to the short memory method. The conventional change-of-basis techniques aim at achieving interoperability between full normal basis implementations and full polynomial basis implementations [7]. In the case of the short memory method, we do not use normal bases for computing field multiplications. In other words, we do not need to use special normal bases such as optimal or Gaussian normal bases. Instead, one might look for normal bases with efficient conversion to polynomial bases.

Table 4. Speed [a] and Memory Comparisons for Software Implementations

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
TNAF_2	543 M	776 M	942 M	1362 M	1901 M
TNAF_3	437 M	620 M	750 M	1079 M	1502 M
	42 bytes	60 bytes	72 bytes	104 bytes	144 bytes
TNAF_4	389 M	541 M	650 M	923 M	1274 M
	126 bytes	180 bytes	216 bytes	312 bytes	432 bytes
TNAF_5	387 M	518 M	612 M	847 M	1150 M
	294 bytes	420 bytes	504 bytes	728 bytes	1008 bytes
Short Memory	395 M	520 M	609 M	834 M	1123 M
(Algorithm 3)	84 bytes	120 bytes	144 bytes	208 bytes	288 bytes

[a] Assumptions for relative costs with multiplication M as reference: squaring $S \approx M/7$, change-of-basis COB $\approx M$, inversion $I \approx 10M$.

6 Conclusion

We proposed a novel technique for computing the scalar multiplication on Koblitz curves. Our method keeps the full power of precomputations without effectively storing all of the precomputed values. In the case of hardware implementations, our method is as fast as the optimal TNAF_5, but reduces memory consumption for precomputations by 85%. In the case of software implementations, again, the running time of our method is roughly the same as the TNAF_5, but with 70% less memory for precomputations. Using our technique, one can

deploy the optimal table size of the TNAF_w without sacrificing much memory to store it. Therefore, for environments with very scarce resources, in software or hardware, the short memory method offers significant improvements compared to known schemes, using the available memory where it is truly needed.

References

1. Avanzi, R. M., Ciet, M., Sica, F.: Faster scalar multiplication on Koblitz curves combining point halving with the Frobenius endomorphism. In Proc. of PKC'04, LNCS 2947, pp.28–40, 2004.
2. Brickell, E. F., Gordon, D. M., McCurley, K. S., Wilson, D. B.: Fast exponentiation with precomputation: algorithms and lower bounds. In Proc. of Eurocrypt'92, LNCS 658, pp.200–209, 1993.
3. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. In IEEE Trans. Computers 53(6), pp. 760–768, 2004.
4. Coron, J.-S., M'Raïhi, D., Tymen, C.: Fast generation of pairs $(k, [k]P)$ for Koblitz elliptic curves. In Proc. of SAC'01, LNCS 2259 , pp. 151–164, 2001.
5. Dahab, R., Hankerson, D., Hu, F., Long, M., López, J., Menezes, A.: Software multiplication using normal bases. Technical report CACR 2004-12, University of Waterloo, 2004.
6. Hankerson, D., López, J., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. In Proc. of CHES'00, LNCS 1965, pp. 1–24, 2001.
7. IEEE P1363A. Standard specifications for public-key cryptography, annex A, number-theoretic background, 2000.
8. Joye, M., Tymen, C.: Compact encoding of non-adjacent forms with applications to elliptic curve cryptography. In Proc. of PKC'01, LNCS 1992, pp. 353–364, 2001.
9. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography: An algebraic approach. In Proc. of CHES'01, LNCS 2162, pp. 377–390, 2001.
10. Kaliski, B.S., Yin, Y.L.: Storage-efficient finite field basis conversion. In Proc. of SAC'98, LNCS 1556, pp. 81–93, 1999.
11. Koblitz, N.: Elliptic curve cryptosystems. In Mathematics of Computation, 48(177), pp. 203–209, 1987.
12. Koblitz, N.: CM-curves with good cryptographic properties. In Proc. of Crypto'91, LNCS 576, pp. 279–287, 1992.
13. Miller, V. S.: Use of elliptic curves in cryptography. In Proc. of Crypto'85, LNCS 218, pp. 417–426, 1986.
14. Massey, J., Omura, J. K.: Computational method and apparatus for finite field arithmetic. US Patent 4587627, 1986.
15. Möller, B.: Improved techniques for fast exponentiation. In Proc. of ICISC'02, LNCS 2587, pp 298-312, 2003.
16. Park, D. J., Sim, S. G., Lee, P. J.: Fast scalar multiplication method using change-of-basis matrix to prevent power analysis attacks on Koblitz curves. In Proc. of WISA 2003, LNCS 2908, pp. 474–488, 2003.
17. Solinas, J. A.: An improved algorithm for arithmetic on a family of elliptic curves. In Proc. of Crypto'97, LNCS 1294, pp. 357–371, 1997.
18. Solinas, J. A.: Efficient arithmetic on Koblitz curves. In Designs, Codes, and Cryptography, 19(2–3), pp. 195–249, 2000.