Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism

Roberto Maria Avanzi[1 ⋆], Mathieu Ciet[2 ⋆], and Francesco Sica[3 ⋆]

[1] IEM, University of Duisburg-Essen, Essen, Germany
mocenigo@exp-math.uni-essen.de
[2] Innova Card, La Ciotat, France – mathieu.ciet@innova-card.com
[3] Dept. of Mathematics and Computer Science, Mount Allison University, Canada
fsica@mta.ca

Dedicated to Preda Mihăilescu
on occasion of the birth of his daughter Seraina.

Abstract. Let E be an elliptic curve defined over $\mathbb{F}_{2^n}$. The inverse operation of point doubling, called point halving, can be done up to three times as fast as doubling. Some authors have therefore proposed to perform a scalar multiplication by an "halve-and-add" algorithm, which is faster than the classical double-and-add method.
If the coefficients of the equation defining the curve lie in a small subfield of $\mathbb{F}_{2^n}$, one can use the Frobenius endomorphism τ of the field extension to replace doublings. Since the cost of τ is negligible if normal bases are used, the scalar multiplication is written in "base τ" and the resulting "τ-and-add" algorithm gives very good performance.
For elliptic Koblitz curves, this work combines the two ideas for the first time to achieve a novel decomposition of the scalar. This gives a new scalar multiplication algorithm which is up to 14.29% faster than the Frobenius method, *without* any additional precomputation.

Keywords. Koblitz curves, scalar multiplication, point halving, τ-adic expansion, integer decomposition.

1 Introduction

In 1985 Miller [9] and Koblitz [7] independently proposed to use the group of rational points of an elliptic curve over a finite field to create cryptosystems based on the discrete logarithm problem (DLP).

The basic operation of a DLP-based cryptosystem is the *scalar multiplication*, i.e. given a point $\boldsymbol{P}$ and an integer s, to compute $s\boldsymbol{P}$. Some

⋆ The European Commission supported the research of the first author under Contract IST-2001-32613 (AREHCC), and the research of the second and third authors under Contract IST-1999-12324 (NESSIE). This research began when the second and third authors were at the UCL Crypto Group, Louvain-la-Neuve, Belgium. The third author's stay at the IEM was supported by the DFG, Graduiertenkolleg 647 *Crypto*.

families of elliptic curves have arithmetic properties useful for speeding up this operation. One such family consists of the *Koblitz curves*: These curves, first proposed by Koblitz [8] and called *anomalous binary curves* by Solinas in [14], are defined over $\mathbb{F}_{2^n}$ by equations of the form

$$E_a \ : \ y^2 + xy = x^3 + ax^2 + 1 \quad \text{with} \quad a \in \{0, 1\} \ . \tag{1}$$

The present paper is devoted to scalar multiplication on Koblitz curves. We restrict our attention to those curves for which n is prime, and whose rational point group contains a (unique) subgroup of large prime order p with a cofactor at most 4, such as those in the standards [17, 18].

Let τ denote the Frobenius endomorphism $\tau(x, y) = (x^2, y^2)$ and $\boldsymbol{P}$ be a point of order p on E_a. As τ commutes with point addition, $\tau(\boldsymbol{P})$ also has order p, and there exists a scalar λ with $\tau(\boldsymbol{P}) = \lambda\boldsymbol{P}$. This suggests that τ may be used to compute multiples of $\boldsymbol{P}$. In fact, we can write a "τ-adic expansion associated to the scalar s", i.e. an expression of the form $\sum_{i=0}^{m} s_i\tau^i$, with $s_i \in \{0, \pm1\}$, such that $\sum_{i=0}^{m} s_i\tau^i(\boldsymbol{P}) = s\boldsymbol{P}$ for all $\boldsymbol{P} \in E_a(\mathbb{F}_{2^n})$. Then a "$\tau$-and-add" loop is used to compute $s\boldsymbol{P}$. Since τ is much faster than a point doubling, the resulting method is very efficient.

Knudsen [5] and Schroeppel [12] independently proposed a technique for elliptic curves over binary fields based on *point halving*. This method computes the multiple $\boldsymbol{R}$ of any point $\boldsymbol{P}$ of odd order such that $2\boldsymbol{R} = \boldsymbol{P}$ and $\boldsymbol{R} \in \langle\boldsymbol{P}\rangle$. Since for curves of order $2p$ point halving is up to three times as fast as doubling, it is possible to improve performance of scalar multiplication by expanding the scalar using "powers of 1/2" and replacing the double-and-add algorithm with a halve-and-add method.

In our paper, we combine for the first time the τ-NAF approach with a single point halving, thereby reducing the amount of point additions from $n/3$ to $2n/7$, and providing an asymptotic speed-up of about 14.29%. *The idea is that it is possible, using a single point halving, to replace some sequences of a τ-NAF having density* $1/2$ *(and containing at least three non-zero coefficients) with sequences having weight* 2.

In the next section we collect some basic facts about τ-NAFs and point halving. In Section 3, we describe our new scalar decomposition, prove its correctness, and apply it to the computation of scalar multiplications. The complexity analysis is given in Section 4. In Section 5 we conclude.

Acknowledgements. The authors express their gratitude to Darrel Hankerson, Tanja Lange, Nicolas Thériault and to the anonymous referees for the many useful suggestions for improving the paper. The authors also thank Jean-Jacques Quisquater for fruitful discussions and support.

2 Background Concepts

2.1 τ Non Adjacent Forms

All facts here are stated without proofs: These are found in [14, 15].

Let the Koblitz curve E_a defined over $\mathbb{F}_{2^n}$ by equation (1) have a (unique) subgroup G of large prime order p with a cofactor at most 4. Let τ denote the Frobenius endomorphism. It is easy to see that for each point $\boldsymbol{P}$ we have $(\tau^2 + 2)\boldsymbol{P} = \mu\, \tau(\boldsymbol{P})$ where $\mu = (-1)^{1-a}$, i.e.

$$\tau^2 + 2 = \mu\tau \ . \tag{2}$$

If τ is identified with a complex root of equation (2), say $\tau = (\mu + \sqrt{-7})/2$, we can view $\tau(\boldsymbol{P})$ as *multiplication by* τ and let $\mathbb{Z}[\tau]$ operate on $\boldsymbol{P}$.

The τ-adic non-adjacent form (τ-NAF for short) of an integer $z \in \mathbb{Z}[\tau]$ is a decomposition $z = \sum_i z_i \tau^i$ where $z_i \in \{0, \pm 1\}$ with the *non-adjacency* property $z_j z_{j+1} = 0$, similarly to the classical NAF [11]. The average *density* (that is the average ratio of non-zero bits related to the total number of bits) of a τ-NAF is 1/3. Each integer z admits a unique τ-NAF. The length of the τ-NAF expansion of a randomly chosen scalar is $\approx 2n$, whereas the bit length of is $\approx n$. But, for any point $\boldsymbol{P} \in E_a(\mathbb{F}_{2^n}) \setminus E_a(\mathbb{F}_2)$, $\tau^n \boldsymbol{P} = \boldsymbol{P}$ and $\tau\boldsymbol{P} \neq \boldsymbol{P}$. Since $\mathbb{Z}[\tau]$ is an Euclidian ring we can take the remainder of $s \bmod (\tau^n - 1)/(\tau - 1)$ and use it in place of s. This remainder will have smaller norm than that of $(\tau^n - 1)/(\tau - 1)$, and thus it will have length at most n. Its τ-NAF is called the *reduced* τ-NAF of s.

The computation of an element of $\mathbb{Z}[\tau]$ of minimal norm which is congruent to s modulo $(\tau^n - 1)/(\tau - 1)$ is a very slow operation. To overcome this problem, Solinas proposes to compute an element which is *almost* of minimal norm and whose computation is much faster. The length of its τ-NAF (the *partially reduced* τ-NAF of s) is at most $n+a+3$. The corresponding τ-and-add algorithm runs marginally slower than with the reduced τ-NAF of the scalar, but the overall speed-up is significant.

2.2 Point Halving

Let E be a generic elliptic curve over $\mathbb{F}_{2^n}$ by an equation of the form

$$E \ : \ y^2 + xy = x^3 + ax^2 + b$$

with $a, b \in \mathbb{F}_{2^n}$ (hence, not necessarily a Koblitz curve) and having a subgroup $G \leq E(\mathbb{F}_{2^n})$ of large prime order. To a point $\boldsymbol{P}$ with affine coordinates (x, y) we associate the quantity $\lambda_{\boldsymbol{P}} = x + \frac{y}{x}$. Let $\boldsymbol{P} = (x, y)$

and $\boldsymbol{R} = (u, v)$ be points of $E(\mathbb{F}_{2^n}) \setminus \{0\}$ with $2\boldsymbol{R} = \boldsymbol{P}$. The affine coordinates of $\boldsymbol{P}$ and $\boldsymbol{R}$ are related as follows:

$$\lambda_{\boldsymbol{R}} = u + \frac{v}{u} \tag{3}$$

$$x = \lambda_{\boldsymbol{R}}^2 + \lambda_{\boldsymbol{R}} + a \tag{4}$$

$$y = u^2 + x(\lambda_{\boldsymbol{R}} + 1) \tag{5}$$

Given $\boldsymbol{P}$, point halving consists in finding $\boldsymbol{R}$. To do this, we have to solve (4) for λ, (5) for u, and finally (3) for v. After some simple manipulations, we see that we have to perform the following operations:

$$\begin{array}{rll} \text{(i)} & \text{Solve } \ \lambda_{\boldsymbol{R}}^2 + \lambda_{\boldsymbol{R}} = a + x \ \text{ for } \ \lambda_{\boldsymbol{R}} & (6) \\ \text{(ii)} & \text{Put } \ t = y + x(\lambda_{\boldsymbol{R}} + 1) & \\ \text{(iii)} & \text{Find } \ u \ \text{ with } \ u^2 = t & (7) \\ \text{(iv)} & \text{Put } \ v = t + u\lambda_{\boldsymbol{R}} \ . & \end{array}$$

Knudsen [5] and Schroeppel [12, 13] show how to perform the necessary steps in an efficient way. A more thorough analysis of the costs of these steps is given in [3]. We shall return to this matter in Section 4.

Point halving is an automorphism of G. So, given a point $\boldsymbol{P} \in G$, there is a unique $\boldsymbol{R} \in G$ such that $2\boldsymbol{R} = \boldsymbol{P}$. In other words, the equations (6) and (7) can always be solved in $\mathbb{F}_{2^n}$. But, they do not determine a *unique* point $\boldsymbol{R}$ with $2\boldsymbol{R} = \boldsymbol{P}$. In fact, solving them will always yield two distinct points $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ such that $\boldsymbol{R}_1 - \boldsymbol{R}_2$ is the unique point of order 2 of the curve. It is possible, by performing an additional check, to determine the point $\boldsymbol{R} \in G$, but we do not need that in our applications. We refer the interested reader to [5, 12, 13] of [3] for details.

3 New Scalar Decomposition and Scalar Multiplication

Consider a Koblitz curve E_a and adopt the notation of Subsection 2.1. Equation (2) implies that $\tau^3 + 2\tau = \mu\tau^2 = \mu(\mu\tau - 2) = \tau - 2\mu$, hence

$$2 = -\mu\big(1 + \tau^2\big)\tau \ . \tag{8}$$

In particular, this means that we can compute $2\boldsymbol{P}$ as $-\mu\big(1 + \tau^2\big)\tau\boldsymbol{P}$. This alone is not very useful, since it replaces a point doubling with one addition and three Frobenius operations. However, these relations become interesting if we can make repeated use of them:

Lemma 1. *Let* $\boldsymbol{P} = 2\boldsymbol{R}$. *Put* $\boldsymbol{Q} = \tau\boldsymbol{R}$. *The following equalities hold:*

$$\Big(\sum_{j=0}^{k-1}(-1)^j\tau^{2j}\Big)\boldsymbol{P} = -\mu(1+(-1)^{k-1}\tau^{2k})\boldsymbol{Q}, \qquad \textbf{(I)}$$

$$\Big(\sum_{j=0}^{k-2}(-1)^j\tau^{2j}\Big)\boldsymbol{P} + (-1)^{k-2}\tau^{2(k-1)}\boldsymbol{P} = (-\mu+(-1)^{k-1}\tau^{2k-1})\boldsymbol{Q}, \quad \textbf{(II)}$$

$$\Big(\sum_{j=0}^{k-3}(-1)^j\tau^{2j}\Big)\boldsymbol{P} + (-1)^{k-3}\big(\tau^{2(k-2)}+\tau^{2(k-1)}\big)\boldsymbol{P} = (-\mu+(-1)^{k-3}\tau^{2k-3})\boldsymbol{Q}. \quad \textbf{(III)}$$

Proof. The first statement is simplified using (8), giving a telescopic sum

$$\sum_{j=0}^{k-1}(-1)^j\tau^{2j}\boldsymbol{P} = -\mu\sum_{j=0}^{k-1}(-1)^j\tau^{2j}(1+\tau^2)\boldsymbol{Q} = -\mu\big(1+(-1)^{k-1}\tau^{2k}\big)\boldsymbol{Q}\ .$$

To prove the second equality we use the previous relation (with $k-1$ in place of k) in combination with the fact that $\boldsymbol{P} = (\mu-\tau)\boldsymbol{Q}$:

$$\begin{aligned}&\Big(\sum_{j=0}^{k-2}(-1)^j\tau^{2j}\Big)\boldsymbol{P} + (-1)^{k-2}\tau^{2(k-1)}\boldsymbol{P} =\\ &\quad = -\mu\big(1+(-1)^{k-2}\tau^{2(k-1)}\big)\boldsymbol{Q} + (-1)^{k-2}\tau^{2(k-1)}(\mu-\tau)\boldsymbol{Q}\\ &\quad = (-\mu+(-1)^{k-1}\tau^{2k-1})\boldsymbol{Q}\ .\end{aligned}$$

The verification of the third equality proceeds in a similar fashion:

$$\begin{aligned}&\Big(\sum_{j=0}^{k-3}(-1)^j\tau^{2j}\Big)\boldsymbol{P} + (-1)^{k-3}\big(\tau^{2(k-2)}+\tau^{2(k-1)}\big)\boldsymbol{P} =\\ &= \big(-\mu+(-1)^{k-2}\tau^{2k-3}\big)\boldsymbol{Q} + (-1)^{k-3}\tau^{2(k-1)}(\mu-\tau)\boldsymbol{Q}\\ &= \big(-\mu+(-1)^{k-2}\tau^{2k-3}(1-\mu\tau+\tau^2)\big)\boldsymbol{Q} = \big(-\mu+(-1)^{k-3}\tau^{2k-3}\big)\boldsymbol{Q}\ .\ \square\end{aligned}$$

We need more terminology and notation to describe and analyze our recoding.

Notation. *We write* $\mathcal{S} = \langle s_n \ldots s_j s_{j-1} \ldots s_1 s_0\rangle$ *for any* τ*-adic expansion (also called string)* $\sum_{0\le j\le n} s_j\tau^j$. *We call* $\#\mathcal{S} = n$ *the* length *of the expansion* $\mathcal{S}$. *Also by* $\mathcal{S}[i\ \ldots\ j]$ *we denote the sub-expansion* $\langle s_i \ldots s_j\rangle$ *of* $\mathcal{S}$. *Occasionally, we will encounter the string* $x \times \langle s_i \ldots s_j\rangle$, *where* $x = \pm 1$. *It is then understood that* $-1 \times \langle s_i \ldots s_j\rangle = \langle -s_i \ldots - s_j\rangle$ *is the bitwise complement of the original string. Henceforth* $\mathcal{S}$ *will denote the* τ*-NAF expansion of any integer, namely an expansion as above with* $s_j = 0, \pm 1$ *and* $s_j s_{j+1} = 0$. *We write* $\bar{1}$ *for* -1, *and also* $\bar{1}^t$ *for* $(-1)^t$.

Definition 1. *Let $\mathcal{K} = \langle \star\, 0 \star \,\ldots\, \star\, 0 \star \rangle$ be a substring of a τ-NAF expansion $\mathcal{S}$, where the symbol $\star$ denotes a 1 or a -1. $\mathcal{K}$ is a* k-block *if it contains k elements $\star$, i.e. it is of length $2k-1$. A k-block is* maximal *if the two digits preceding it and the two following it are all zero.*

Example 1. We highlight a few examples of k-blocks in a sequence

$$\langle 1\,0\,0\,\underbrace{\overbrace{1\,0\,\bar{1}}^{\text{2-block}}\,0\,1}_{\substack{\text{(maximal)}\\ \text{3-block}}}\,0\,0\,0\,1\,0\,0\,\underbrace{\bar{1}\,0\,\overbrace{\bar{1}\,0\,\bar{1}\,0\,1}^{\text{3-block}}}_{\substack{\text{(maximal)}\\ \text{4-block}}}\,0\,0\,\bar{1}\rangle\ .$$

We now give a practical application of Lemma 1.

Remark 1. Let s be an integer and $\boldsymbol{P}$ a point of odd order on a Koblitz curve. Let $\mathcal{S} = \langle s_{\ell-1} \ldots s_j s_{j-1} \ldots s_1 s_0 \rangle$ be the τ-NAF associated to s, so that $s\boldsymbol{P} = \sum_{j=0}^{\ell-1} s_j \tau^j(\boldsymbol{P})$. By Lemma 1, the multiples of $\boldsymbol{P}$ corresponding to some special k-blocks appearing in $\mathcal{S}$ can be computed as suitable multiples of $\boldsymbol{Q} := \tau\left(\frac{1}{2}\boldsymbol{P}\right)$ by a τ-and-add method involving fewer group additions. The situation, in terms of substrings of τ-adic expansions, is the following (where all blocks on the left-hand side are k-blocks).

$$\langle \underbrace{\bar{1}^{k-1}\,0\,\bar{1}^{k-2}\,0 \quad \ldots \quad 0\,1\,0\,\bar{1}\,0\,1}_{\text{length } 2k-1} \rangle \boldsymbol{P} = \bar{\mu} \langle \underbrace{\bar{1}^{k-1}\,0\,0\ \ldots\ 0\,0\,1}_{\text{length } 2k+1} \rangle \boldsymbol{Q} \tag{I}$$

$$\langle \underbrace{\bar{1}^{k-2}\,0\,\bar{1}^{k-2}\,0\,\bar{1}^{k-3}\,0 \quad \ldots \quad 0\,1\,0\,\bar{1}\,0\,1}_{\text{length } 2k-1} \rangle \boldsymbol{P} = \langle \underbrace{\bar{1}^{k-1}\,0\,0 \ldots 0\,0\,\bar{\mu}}_{\text{length } 2k} \rangle \boldsymbol{Q} \tag{II}$$

$$\langle \underbrace{\bar{1}^{k-3}\,0\,\bar{1}^{k-3}\,0\,\bar{1}^{k-3}\,0\,\bar{1}^{k-4}\,0\ \ldots 0\,1\,0\,\bar{1}\,0\,1}_{\text{length } 2k-1} \rangle \boldsymbol{P} = \langle \underbrace{\bar{1}^{k-3}\,0\,0 \ldots 0\,\bar{\mu}}_{\text{length } 2k-2} \rangle \boldsymbol{Q}. \tag{III}$$

Definition 2. *We call the k-blocks of the above three types together with their opposites in sign* good *k-blocks. A* maximal *good k-block is a good k-block which cannot be further extended at its sides.*

Remark 1 suggests a strategy for saving operations in the computation of $s\boldsymbol{P}$. From the τ-NAF $\mathcal{S}$ of s, we create *two* τ-adic expansions, $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, by repeated replacements of subsequences, where:

1. $\mathcal{S}^{(1)}$ is obtained from $\mathcal{S}$ by discarding the maximal good k-blocks for $k \geq 3$, substituting them with a string of $2k-1$ zeros;
2. $\mathcal{S}^{(2)}$ consists of the weight two right-hand sequences replacing the maximal good k-blocks removed from $\mathcal{S}$, each at the same position where the original k-block was in $\mathcal{S}$, according to **I**, **II** or **III**.

Input: A Koblitz curve E_a with corresponding parameter $\mu = (-1)^{1-a}$, a point $\boldsymbol{P}$ of odd order on E_a and a scalar s with associated (partially) reduced τ-NAF $\mathcal{S}$
Output: Two τ-adic expansions $\mathcal{S}^{(j)} = \sum_i s_i^{(j)} \tau^i$, $j = 1, 2$ such that $s\boldsymbol{P} = \mathcal{S}^{(1)}\boldsymbol{P} + \mathcal{S}^{(2)}\boldsymbol{Q}$, where $\boldsymbol{Q} = \tau(\frac{1}{2}\boldsymbol{P})$

```
S^(1) ← S, S^(2) ← ⟨0 ... 0⟩ with #S^(2) = #S + 2, and i ← 0
DO {
  x ← s_i
  If x = 0 then { i ← i + 1 }
  else {
    Let k ≥ 1 be the largest integer such that:
        S[i + 2(k − 1) ... i] = x × ⟨ 1̄^(k−1) 0 1̄^(k−2) 0 ... 1̄ 0 1 ⟩
    type ← I
    If s_{i+2k} = s_{i+2(k−1)} then { k ← k + 1 and type ← II ,
      If s_{i+2k} = s_{i+2(k−1)} then { k ← k + 1 and type ← III } }
    (Observe that s_{i+2k−1} = 0)
    If k ≥ 3 then {
      S^(1)[i + 2(k − 1) ... i] ← ⟨0 ... 0⟩
      If type = I   then { s^(2)_{i+2k} ← (−1)^k μx and s^(2)_i ← −μx }
      If type = II  then { s^(2)_{i+2k−1} ← (−1)^(k−1) x and s^(2)_i ← −μx }
      If type = III then { s^(2)_{i+2k−3} ← (−1)^(k−3) x and s^(2)_i ← −μx }
    }
    i ← i + 2k
  }
} WHILE i ≤ #S
Output (S^(1), S^(2)).
```

Algorithm 1. New τ-adic scalar recoding

It is clear from Lemma 1 and Remark 1 that $s\boldsymbol{P} = \mathcal{S}^{(1)}\boldsymbol{P} + \mathcal{S}^{(2)}\boldsymbol{Q}$.

Remark 2. It is easy to verify that no two k-block replacements overlap. For k-blocks of types **II** and **III** this is obvious. Since a maximal k-block of type **I** is followed by at least *two* zero bits (otherwise it would not be maximal), the next non-zero bit may only occur after the end of the replacement block. $\mathcal{S}^{(2)}$ need not satisfy the non-adjacency property.

We have written down explicity the algorithm which generates $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$ as Algorithm 1. Note that the length of $\mathcal{S}^{(1)}$ is equal to the length of $\mathcal{S}$ and that of $\mathcal{S}^{(2)}$ is at most the length of $\mathcal{S}$ plus two.

The total number of non-zero coefficients in $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$ is, by construction, no greater than that of $\mathcal{S}$. In fact, the number of non-zero coefficients decreases considerably on average (see Section 4). We now see how to use the new recoding to perform a scalar multiplication.

3.1 Field represented using a normal basis.

If n is prime, then a normal basis for $\mathbb{F}_{2^n}$ exists and it is easy to construct [1]. Squaring an element of the field consists in a circular shift of the bits of the internal representation of its argument. The same holds for the inverse operation, the extraction of a square root. Therefore, τ, and its inverse, have the same minimal cost.

To compute $\mathcal{S}^{(1)}\boldsymbol{P} + \mathcal{S}^{(2)}\boldsymbol{Q}$, it is not necessary to precompute $\boldsymbol{Q}$: We can first compute $\mathcal{S}^{(2)}\boldsymbol{P}$, halve the result, apply a suitable power of τ, and then resume the τ-and-add loop using $\mathcal{S}^{(1)}$, thus avoiding an extra point storage. We give a realization of this idea which processes the τ-adic expansions right-to-left (i.e. beginning with the lowest powers of τ) and using τ^{-1} instead of τ. In Remark 3 we will see how this allows to interleave our recoding of $\mathcal{S}$ into $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$ with the scalar multiplication.

We begin by computing $\mathcal{S}^{(2)}\boldsymbol{P}$. We first set a variable $\boldsymbol{X}$ to $s_0^{(2)}\boldsymbol{P}$. For each $j = 1, 2, \ldots, \ell_2 - 1$ with $\ell_2 = \#\mathcal{S}^{(2)}$ we apply τ^{-1} to $\boldsymbol{X}$ and add $s_j^{(2)}\boldsymbol{P}$. After these steps $\boldsymbol{X}$ equals $\tau^{-\ell_2+1}\mathcal{S}^{(2)}\boldsymbol{P}$ because we used the exponentiation algorithm from *right to left* with τ^{-1} instead of τ, so we apply τ^{ℓ_2-1-n} to get the correct result. (We use the fact that $\tau^n - 1$ is 0 on E.) We then replace $\boldsymbol{X}$ with $\tau\left(\frac{1}{2}\boldsymbol{X}\right)$ and repeat the above procedure with $\mathcal{S}^{(1)}$ in place of $\mathcal{S}^{(2)}$, starting from $\boldsymbol{X} + s_0^{(1)}\boldsymbol{P}$. We have thus Algorithm 2.

Remark 3. Once the τ-NAF $\mathcal{S}$ is given, there is no need to store $\mathcal{S}^{(j)}$ for $j = 1, 2$. The generation of $\mathcal{S}^{(j)}$ for $j = 1, 2$ can be done twice and online, during the run of Algorithm 2. For simplicity we do not write down the resulting algorithm. The result is: *The scalar multiplication algorithm based on the new scalar decomposition can be done without any precomputations, and without requiring storage for the recoding.*

3.2 Field represented using a polynomial basis.

In this case, squarings have a small, yet non-negligible cost: According to the experiments in [4, Section 3.5] we can assume $\frac{S}{M} \approx \frac{1}{8}$ for $n = 163$ and $\frac{S}{M} \approx \frac{1}{10}$ for $n = 233$. Knudsen [5] expects *"the time to compute a square root in a polynomial basis to be equivalent to half the time to compute a field multiplication plus a very small overhead"*. This is in the general case confirmed in [3]. So, τ and τ^{-1} have in general different costs. In [3] a special square root extraction algorithm is given if the field is represented via a trinomial: in the case of $\mathbb{F}_{2^{233}}$, a good trinomial is $f(x) = x^{233} + x^{74} + 1$ and a square root costs about $\frac{1}{8}M$.

If we use Algorithm 2 to perform a scalar multiplication, we pay a penalty due to the increased number of Frobenius (τ^{-1}) operations. One

Input: A Koblitz curve E_a with corresponding parameter $\mu = (-1)^{1-a}$, a point $\boldsymbol{P}$ of odd order on E_a and a scalar s with associated (partially) reduced τ-NAF $\mathcal{S}$
Output: $s\boldsymbol{P}$

Compute the two τ-adic expansions
$\mathcal{S}^{(j)} = \sum_{i=0}^{\ell_j - 1} s_i^{(j)} \tau^i$ for $j = 1, 2$
from $\mathcal{S}$ using Algorithm 1
(If $\mathcal{S}$ is the reduced τ-NAF of s then $\#\mathcal{S}$ and $\ell_1 \le n$.
If $\mathcal{S}$ is partially reduced then $\#\mathcal{S}$, $\ell_1 \le n + a + 3$.
ℓ_2 is at most $\#\mathcal{S} + 2$.)
$\boldsymbol{X} \leftarrow s_0^{(2)} \boldsymbol{P}$
for $j = 1$ to $\ell_2 - 1$ do
{ $\boldsymbol{X} \leftarrow \tau^{-1}\boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(2)}\boldsymbol{P}$ }
(Now $\boldsymbol{X} = \tau^{-\ell_2+1}\mathcal{S}^{(2)}\boldsymbol{P}$)
$\boldsymbol{X} \leftarrow \tau^{\ell_2 - n}\boldsymbol{X}$, $\boldsymbol{X} \leftarrow \frac{1}{2}\boldsymbol{X}$
(Here we simplified $\boldsymbol{X} \leftarrow \tau^{\ell_2-1-n}\boldsymbol{X}$, $\boldsymbol{X} \leftarrow \tau\left(\frac{1}{2}\boldsymbol{X}\right)$.
Now $\boldsymbol{X} = \mathcal{S}^{(2)}\tau\left(\frac{1}{2}\boldsymbol{P}\right)$.)
$\boldsymbol{X} \leftarrow \boldsymbol{X} + s_0^{(1)}\boldsymbol{P}$
for $j = 1$ to $\ell_1 - 1$ do
{ $\boldsymbol{X} \leftarrow \tau^{-1}\boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(1)}\boldsymbol{P}$ }
(Now $s = \tau^{-\ell_1+1}\left(\mathcal{S}^{(1)}\boldsymbol{P} + \mathcal{S}^{(2)}\tau\left(\frac{1}{2}\boldsymbol{P}\right)\right) = \tau^{-\ell_1+1}s\boldsymbol{P}$)
$\boldsymbol{X} \leftarrow \tau^{\ell_1 - 1 - n}\boldsymbol{X}$
Output ($\boldsymbol{X}$).

Algorithm 2. New scalar multiplication algorithm, right–to–left

way to overcome this problem is to compute $\mathcal{S}^{(1)}\boldsymbol{P} + \mathcal{S}^{(2)}\boldsymbol{Q}$ using the *joint* representation obtained from $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, i.e. the sequence of pairs $\left(s_i^{(1)}, s_i^{(2)}\right)_{i \ge 0}$ and Shamir's trick (actually due to Straus [16] and in a more general form). By Remark 2, at most one element in each pair $\left(s_i^{(1)}, s_i^{(2)}\right)$ is non-zero: Hence, we can use the Straus-Shamir trick without the need to precompute $\boldsymbol{P} \pm \boldsymbol{Q}$, and we only need to store $\boldsymbol{Q}$.

A better solution when the extraction of square roots is (relatively) expensive is to use a variant of Algorithm 2 with τ instead of τ^{-1}. We write it down as Algorithm 3: In this case we must store the τ-adic expansion before the scalar multiplication, and we need to compute and store each of $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, before the corresponding τ-and-add loop.

4 Analysis and Performance Aspects

In the next subsection we prove the reduction of 14.29% in group additions of our method with respect to the τ-and-add method based on the τ-NAF. In Subsection 4.2 we estimate the effective improvement brought by our techniques by considering all group operations.

Input: A Koblitz curve E_a with corresponding parameter $\mu = (-1)^{1-a}$, a point $\boldsymbol{P}$ of odd order on E_a and a scalar s with associated (partially) reduced τ-NAF $\mathcal{S}$
Output: $s\boldsymbol{P}$

Compute the two τ-adic expansions
$\qquad \mathcal{S}^{(j)} = \sum_{i=0}^{\ell_j - 1} s_i^{(j)} \tau^i$ for $j = 1, 2$
from $\mathcal{S}$ using Algorithm 1
$\boldsymbol{X} \leftarrow s_{\ell_2 - 1}^{(2)} \boldsymbol{P}$
for $j = \ell_2 - 2$ to 0 do
$\quad \{ \boldsymbol{X} \leftarrow \tau \boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(2)} \boldsymbol{P} \}$
$\qquad$(Now $\boldsymbol{X} = \mathcal{S}^{(2)} \boldsymbol{P}$)
$\boldsymbol{X} \leftarrow \tau^{n+2-\ell_1} \boldsymbol{X}$, $\boldsymbol{X} \leftarrow \frac{1}{2} \boldsymbol{X}$
$\qquad$(Here we simplified $\boldsymbol{X} \leftarrow \tau^{-\ell_1+1+n} \boldsymbol{X}$, $\boldsymbol{X} \leftarrow \tau\left(\frac{1}{2}\boldsymbol{X}\right)$.
$\qquad$ Now $\boldsymbol{X} = \mathcal{S}^{(2)} \tau^{-\ell_1+2}\left(\frac{1}{2}\boldsymbol{P}\right)$.)
$\boldsymbol{X} \leftarrow \boldsymbol{X} + s_{\ell_1 - 1}^{(1)} \boldsymbol{P}$
for $j = \ell_1 - 2$ to 0 do
$\quad \{ \boldsymbol{X} \leftarrow \tau \boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(1)} \boldsymbol{P} \}$
$\qquad$(Now $s = \tau^{\ell_1 - 1} \mathcal{S}^{(2)} \tau^{-\ell_1+2}\left(\frac{1}{2}\boldsymbol{P}\right) + \mathcal{S}^{(1)} \boldsymbol{P} = \mathcal{S}^{(1)} \boldsymbol{P} + \mathcal{S}^{(2)} \boldsymbol{Q}$.)
Output $(\boldsymbol{X})$.

Algorithm 3. New scalar multiplication algorithm, left–to–right

4.1 Complexity analysis

The following lemma can be proved analysing the τ-NAF recoding algorithm. Similar results hold for the usual NAF (see for example [2]).

Lemma 2. *In a τ-NAF the probability that the digit immediately to the left of a 0 is another 0 is $\frac{1}{2}$ and that it is 1 or -1 is $\frac{1}{4}$ in each case*(i).

To prove that our method gives an expected 14.29% reduction in group additions over the classical τ-and-add method, we model the reading of $\mathcal{S}$ in Algorithm 1 – and the consequent construction of $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$ – in terms of Markov chains. To do this, we describe the algorithm as a sequence of states taken from a list $\{\Sigma_0, \ldots, \Sigma_r\}$. States $\Sigma_0, \ldots, \Sigma_r$ occur with respective *limiting probabilities* $\sigma_0, \ldots, \sigma_r$. The states must be subject to the condition that the probability π_{ij} that the state following Σ_i is Σ_j depends only on the States Σ_i and Σ_j and not on the way State Σ_i has been reached. If $\Pi = (\pi_{ij})_{i,j=0}^{r}$ then the probabilities $\sigma_0, \ldots, \sigma_r$ sum up to 1 and form a vector $\boldsymbol{\sigma} = (\sigma_0 \ \ldots \ \sigma_r)$ such that $\boldsymbol{\sigma}\Pi = \boldsymbol{\sigma}$.

While scanning $\mathcal{S}$ in Algorithm 1 we are either attempting to form a maximal good k-block, or skipping zeros between blocks. We define five different states.

(i) The given probabilities are actually correct up to an error term exponentially decreasing in the length of the τ-NAF, and that does not influence the following analysis significantly.

Σ_0: The state in which zeros outside k-blocks are skipped. Only one zero is skipped. All other states describe operations done to build k-blocks.

Σ_1: Entered whenever the first non-zero bit in a k-block is found. This is the one and only state where the first non-zero bit of a new k-block is read. Of course a zero bit follows and is skipped (the same also holds for States Σ_2–Σ_4). The following three states describe the scanning of the next bits in the k-block begun by entering State Σ_1.

Σ_2: Entered every time we find a non-zero bit which is the negative of the previous non-zero bit read. It can only follow States Σ_1 or Σ_2 itself.

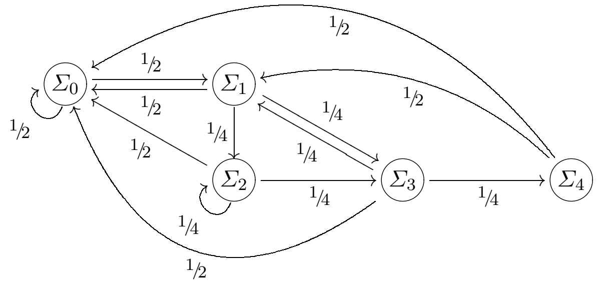
Σ_3: This state corresponds to the *first* non-zero bit having the same sign as the previous one. Either this bit is the last non-zero bit in a type **II** k-block or the second to last in a type **III** k-block.

Σ_4: Entered after Σ_3 if the third in a line of three non-zero bits having the same sign is found. This bit is the last bit in a type **III** k-block.

State Σ_0 is reached if and only if the bit to the left of the bit(s) of the previous state is 0. We recall that in all states except Σ_0 the algorithm actually processes *two* bits: a non-zero bit whose relation to the previous non-zero bits determines the actual state, and the following zero.

State Σ_1 may follow States Σ_3 and Σ_4 directly. This occurs when a k-block follows immediately a maximal good k-block of type **II** or **III**.

The following state diagram illustrates the flow of the algorithm. The nodes correspond to the states and the arrows are labelled with the transition probabilities, which follow immediately from Lemma 2.



Recall that π_{ij} denotes the transition probability from state Σ_i to state Σ_j. We have the following *probability transition matrix*:

$$\Pi = (\pi_{ij})_{i,j=0}^{4} = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/4 & 1/4 & 0 \\ 1/2 & 0 & 1/4 & 1/4 & 0 \\ 1/2 & 1/4 & 0 & 0 & 1/4 \\ 1/2 & 1/2 & 0 & 0 & 0 \end{pmatrix}.$$

Now that Π is known, we can easily compute the limiting probabilities $\sigma_0, \ldots, \sigma_4$, which are uniquely determined, and are: $\boldsymbol{\sigma} = \frac{1}{42}(\, 21\ 12\ 4\ 4\ 1\,)$.

Now suppose that, after λ state transitions, the algorithm has processed m bits of $\mathcal{S}$ and output a total of w non-zero bits in $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$. Since in state Σ_0 only one bit of $\mathcal{S}$ is scanned and in all other states two, after λ state transitions the expected number of processed bits is $m = \lambda(\sigma_0 + 2(1-\sigma_0)) = \lambda\big(\frac{1}{2} + 2\cdot\frac{1}{2}\big) = \frac{3}{2}\lambda$.

Now, good k-blocks of weight 1 and 2 are left in $\mathcal{S}^{(1)}$, whereas good k-blocks of weight at least 3 are cleared from $\mathcal{S}^{(1)}$ and appropriate sequences of weight 2 are inserted in $\mathcal{S}^{(2)}$ as described in Algorithm 1. Suppose the algorithm enters State Σ_1. If it immediately goes to State Σ_0, only one non-zero bit is output. In all other cases two non-zero bits are output. Then $w = \sigma_1\lambda\big(1\cdot\pi_{10} + 2\cdot(1-\pi_{10})\big) = \frac{12}{42}\lambda\big(\frac{1}{2} + 2\cdot\frac{1}{2}\big) = \frac{3}{7}\lambda$.

Last, suppose the length of the original τ-NAF is m. It has, as already recalled, about $m/3$ non-zero digits. However the number of the non-zero digits in $\mathcal{S}^{(1)} \cup \mathcal{S}^{(2)}$ is $2m/7$. Since the number of additions equals the number of non-zero digits, minus one, our method brings a reduction of $\big(\frac{1}{3} - \frac{2}{7}\big)/\frac{1}{3} \approx 14.29\%$ in additions with respect to the τ-and-add method.

4.2 Practical estimates

We now estimate the actual speed-up for specific curves. As examples, we shall consider the Koblitz curves K-163 and K-233 over $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_{2^{233}}$ from the FIPS standard issued by NIST [18].

Point halving (H), as described in Subsection 2.2, requires two field multiplications (M), the solution of an equation in λ of the type $\lambda^2+\lambda = c$ (EQ) and the extraction of a square root ($\sqrt{\ }$). An elliptic curve addition (A) is done by one field inversion (I), two multiplications and one squaring (S). A point doubling (D) requires $I + 2\,M + 2\,S$. A Frobenius operation (τ) and its inverse (τ^{-1}) require $2\,S$ and $2\,\sqrt{\ }$ respectively.

With a polynomial basis, according to [4], $S \approx \frac{1}{7.5}M$ for $n = 163$ and $\frac{1}{9}M$ for $n = 233$. Following [3] we assume that, on average, $I \approx 8\,M$ when $n = 163$ and $I \approx 10\,M$ when $n = 233$. (For a comparison, [10] has $I \approx 9.3\,M$ for $n = 191$, for a software implementation on an embedded processor.) In $\mathbb{F}_{2^{233}}$, a field defined by a trinomial, a square root can be computed in $\approx \frac{1}{8}M$ [3, Example 3.12]. For $\mathbb{F}_{2^{163}}$ only a generic method is currently known, so $\sqrt{\ } \approx \frac{1}{2}M$. EQ takes, experimentally $\approx \frac{2}{3}M$.

If a normal basis is used, [5], S, $\sqrt{\ }$ and EQ have negligible costs. Because of the relatively high cost of a multiplication, we may assume $I \approx 3\,M$.

Since the length of a τ-expansion is $\approx n+a+3$ (see Subsection 2.1), we see that the expected cost of the τ-and-add algorithm is $\frac{1}{3}(n+a+2)A+(n+a+2)\tau$. Algorithm 2 requires $\frac{2}{7}(n+a+2)A+2(n+a+2)\tau^{-1}$ in the two loops; *Between* the two loops there are: H, $1\,A$, and on average $(n+a+3)-n=a+3$ Frobenius operations (τ). Algorithm 3 has similar costs in the main loops, with τ in place of τ^{-1}, but, on average, between the loops there is only a doubling and one addition. If the Straus-Shamir method is used (with a polynomial basis) right-to-left and with a single precomputation, the cost is $\frac{2}{7}(n+a+2)A+(n+a+3)\tau+H$.

In the following table we write down the costs of different scalar multiplication algorithms relative to that of one multiplication: the τ-and-add method based on the τ-NAF, our Algorithms 2 and 3 with the gain of the bast of the latter two over the τ-and-add. In the case of polynomial basis, we also show the costs of two methods requiring one precomputation: the one based on the Straus-Shamir trick from Subsection 3.2, and the usage of the width-2 τ-NAF (see [14, 15]), which needs only $3\boldsymbol{P}$.

n	a	basis	τ-&-A	Algo. 2	Algo. 3	gain w.r.t. τ-&-A	*width-2 τ-&-A*	*Straus- -Shamir*
163	1	NB	276.7	244.1	–	11.8 %	–	–
		poly	605	827	572.4	5.5 %	*485.2*	*528.3*
233	0	NB	391.7	342.7	–	12.5 %	–	–
		poly	1001	946.2	932.5	7 %	*788.1*	*868.4*

The speed-ups are less than the theoretical estimate because of the additional overheads. The improvements will approach the theoretical maximum for large n. Our estimates are for software implementations. In hardware, where the ratio I/M is higher, the actual improvement will be much closer to the asymptotic maximum. But in that case one should also consider the use of projective coordinates. If one can store one precomputed point, the width-2 τ-NAF is faster than the Straus-Shamir trick.

5 Conclusions

In this paper we considered the problem of computing scalar multiplications on Koblitz curves. We combined for the first time the τ-adic expansion with point halving to give a new recoding of the scalar. By means of this we reduced the number of group operations required for a scalar multiplication by an asymptotic 14.29%.

For the curves K-163 and K-233 from NIST's FIPS standard we estimate an overall speedup of at least 12% if a normal basis is used.

The case where the field extension is represented using a normal basis is of particular relevance. It gives the highest speed-up, it allows to perform the scalar recoding online in the scalar multiplication, hence has no additional memory requirements (with respect to the classical τ-and-add method), apart from code size.

References

1. D. W. Ash, I. F. Blake and S. Vanstone. *Low complexity normal bases.* Discrete Applied Math. **25** (1989), pp. 191–210.
2. R. M. Avanzi. *On the complexity of certain multi-exponentiation techniques in cryptography.* To appear in Journal of Cryptology.
3. K. Fong, D. Hankerson, J. Lopez and A. Menezes. *Field inversion and point halving revisited.* Available from `http://www.cs.siu.edu/~kfong/research/ECCpaper.ps`, Unpublished Manuscript.
4. D. Hankerson, J. Lopez-Hernandez, and A. Menezes. *Software Implementatin of Elliptic Curve Cryprography over Binary Fields.* In: *Proceedings of CHES 2000.* LNCS 1965, pp. 1–24. Springer, 2001.
5. E. W. Knudsen. *Elliptic Scalar Multiplication Using Point Halving.* In: *Proocedings of ASIACRYPT 1999*, LNCS 1716, pp. 135–149. Springer, 1999.
6. D. E. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1999. 3rd ed.
7. N. Koblitz. *Elliptic curve cryptosystems.* Mathematics of computation **48** (1987), pp. 203–209.
8. N. Koblitz. *CM-curves with good cryptographic properties.* In: *Proceedings of CRYPTO 1991*, LNCS 576, pp. 279–287. Springer, 1991.
9. V. S. Miller. *Use of elliptic curves in cryptography.* In: *Proceedings of CRYPTO '85.* LNCS 218, pp. 417–426. Springer, 1986.
10. J. Pelzl, T. Wollinger, J. Guajardo and C. Paar. *Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves.* In: *Proceedings of CHES 2003.* LNCS 2779, pp. 351–365. Springer 2003.
11. G. W. Reitwiesner. *Binary arithmetic.* Advances in Computers, 1:231–308, 1960.
12. R. Schroeppel. *Point halving wins big.* Talks at: (i) Midwest Arithmetical Geometry in Cryptography Workshop, November 17–19, 2000, University of Illinois at Urbana-Champaign; and (ii) ECC 2001 Workshop, October 29–31, 2001, University of Waterloo, Ontario, Canada.
13. R. Schroeppel. *Elliptic curve point ambiguity resolution apparatus and method.* International Application Number PCT/US00/31014, filed 9 November 2000.
14. J. A. Solinas. *An improved algorithm for arithmetic on a family of elliptic curves.* In: *Proceedings of CRYPTO 1997*, LNCS 1294, pp. 357–371. Springer, 1997.
15. J. A. Solinas. *Efficient Arithmetic on Koblitz Curves.* Designs, Codes and Cryptography, Vol. 19 (2000), No. 2/3, pp. 125–179.
16. E. G. Straus, *Addition chains of vectors (problem 5125).* American Mathematical Monthly, vol. 71, 1964, pp. 806–808.
17. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography.* IEEE Computer Society, August 29, 2000.
18. National Institute of Standards and Technology. *Digital Signature Standard.* FIPS Publication 186-2, February 2000.