How to Use Koblitz Curves on Small Devices?

Kimmo Järvinen[1,2] and Ingrid Verbauwhede[1]

[1] KU Leuven ESAT/COSIC and iMinds
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
[2] Aalto University, Department of Information and Computer Science
Konemiehentie 2, FI-02150 Espoo, Finland
firstname.lastname@esat.kuleuven.be

Abstract. Koblitz curves allow very efficient scalar multiplications because point doublings can be traded for cheap Frobenius endomorphisms by representing the scalar as a τ-adic expansion. Typically elliptic curve cryptosystems, such as ECDSA, also require the scalar as an integer. This results in a need for conversions between integers and the τ-adic domain, which are costly and prevent from using Koblitz curves on very constrained devices, such as RFID tags or wireless sensors. In this paper, we provide a solution to this problem by showing how complete cryptographic processes, such as ECDSA signing, can be completed in the τ-adic domain with very few resources, consequently outsourcing the expensive conversions to a more powerful party. We also provide small circuitries that require about 76 gate equivalents on 0.13 μm CMOS and that are applicable for all Koblitz curves.

1 Introduction

Because elliptic curve cryptography (ECC) [12, 18] offers high security levels with short key lengths and relatively low amounts of computation, it is one of the most feasible alternatives for implementing public-key cryptography on constrained devices where resources (e.g., circuit area, power, and energy) are extremely limited. Constrained devices that require lightweight implementations of public-key cryptography are, e.g., wireless sensor network nodes, RFID tags, and smart cards. Several researchers have proposed lightweight implementations which aim to minimize area, power, and/or energy of computing elliptic curve scalar multiplications [2, 3, 8, 14, 16], which are the fundamental operations required by every elliptic curve cryptosystem.

Koblitz curves [13] are a special class of elliptic curves offering very efficient elliptic curve operations when scalars used in scalar multiplications are given as τ-adic expansions. It is commonly known that Koblitz curves allow extremely fast scalar multiplications in both software [25] and hardware [9]. A recent paper [2] showed that they can be implemented also with very few resources (especially, in terms of energy) if the scalar is already in the τ-adic domain. Many cryptosystems require both the integer and τ-adic representations of the scalar which results in a need for conversions between the domains. All known methods for computing these conversions in hardware [1, 5, 6, 10, 23] require a lot of

resources making them unfeasible for constrained devices. In most cases, this prevents from using Koblitz curves although they would otherwise result in very efficient lightweight implementations. A workaround to this problem is to design a protocol that operates directly in the τ-adic domain [4]. However, this approach has several drawbacks because it prevents from using standardized algorithms and protocols, which, consequently, makes the design work more laborious and may even lead to cryptographic weaknesses in the worst case.

In this paper, we show how the computationally weaker party of a cryptosystem can delegate the conversions to the more powerful party by computing all operations directly in the τ-adic domain with an extremely small circuitry. Our approach can be straightforwardly used for many existing Koblitz curve cryptosystems that require scalar multiplications on Koblitz curves and modular arithmetic with the scalar (e.g., ECDSA) without affecting the cryptographic strength of the cryptosystem. We also provide small circuitries that enable efficient lightweight implementations of the approach. Consequently, we show how Koblitz curves can be used also in lightweight implementations.

This paper is structured as follows. Section 2 surveys the preliminaries of ECC and Koblitz curves. Section 3 discusses the existing solutions for using Koblitz curves by reviewing the related work on conversions between integers and the τ-adic domain and presents the outline of the new idea. An algorithm for computing additions in the τ-adic domain is presented and analyzed in Section 4. Section 5 presents algorithms for computing other arithmetic operations using the algorithm from Section 4. Section 6 introduces an architecture for a circuitry implementing the algorithms from Sections 4 and 5. Section 7 presents implementation results on 0.13 μm CMOS and compares them to converters from the literature. Section 8 presents a case study of how the findings of this paper could be used in computing ECDSA signatures in lightweight implementations. The paper ends with conclusions in Section 9.

2 Elliptic Curve Cryptography and Koblitz Curves

In the mid-1980s, Miller [18] and Koblitz [12] showed how public-key cryptography can be based on the difficulty of computing the discrete logarithm in an additive Abelian group $\mathcal{E}$ formed by points on an elliptic curve. Let $k \in \mathbb{Z}_+$ and $\mathbf{P} \in \mathcal{E}$. The fundamental operation in ECC is the scalar multiplication which is given by:

$$k\mathbf{P} = \underbrace{\mathbf{P} + \mathbf{P} + \ldots + \mathbf{P}}_{k \text{ times}} \ . \tag{1}$$

The operation $\mathbf{Q} + \mathbf{R}$, where $\mathbf{Q}, \mathbf{R} \in \mathcal{E}$, is called point addition if $\mathbf{Q} \neq \pm\mathbf{R}$ and point doubling if $\mathbf{Q} = \mathbf{R}$. Scalar multiplication can be computed with a series of point doublings and point additions, e.g., by using the well-known double-and-add algorithm. Elliptic curves over finite fields of characteristic two $GF(2^m)$ are often preferred in implementing ECC because they allow efficient implementations, especially, in hardware. These curves are commonly called binary curves.

Koblitz curves [13] are a subclass of binary curves defined by the equation:

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

where $a \in \{0, 1\}$ and $x, y \in GF(2^m)$. Let $\mathcal{K}$ denote the Abelian group of points (x, y) that satisfy (2) together with $\mathcal{O}$ which is a special point that acts as the zero element of the group. Koblitz curves have the property that if a point $\mathbf{P} = (x, y) \in \mathcal{K}$, then also its Frobenius endomorphism $F(\mathbf{P}) = (x^2, y^2) \in \mathcal{K}$. This allows devising efficient scalar multiplication algorithms where Frobenius endomorphisms are computed instead of point doublings. It can be shown that $F(F(\mathbf{P})) - \mu F(\mathbf{P}) + 2\mathbf{P} = 0$, where $\mu = (-1)^{1-a}$, holds for all $\mathbf{P} \in \mathcal{K}$ [13]. Consequently, $F(\mathbf{P})$ can be seen as a multiplication by the complex number τ that satisfies $\tau^2 - \mu\tau + 2 = 0$, which gives $\tau = (\mu + \sqrt{-7})/2$.

If the scalar k is given using the base τ as a τ-adic expansion $K = \sum K_i \tau^i$, the scalar multiplication $K\mathbf{P}$ can be computed with a Frobenius-and-add algorithm, where Frobenius endomorphisms are computed for each K_i and point additions (or subtractions) are computed for $K_i \neq 0$. This is similar to the double-and-add algorithm except that computationally expensive point doublings are replaced with cheap Frobenius endomorphisms. Hence, if a τ-adic expansion can be efficiently found, then Koblitz curves offer considerably more efficient scalar multiplications than general binary curves.

We use the following notation. Lower-case letters $a, b, c, \ldots$ denote integer values and upper-case letters $A, B, C, \ldots$ denote τ-adic expansions. If both lower-case and upper-case version of the same letter are used in the same context, then the values are related; to state this explicitly, we denote $A \stackrel{\circ}{=} a$. Bold-faced upper case letters $\mathbf{P}, \mathbf{Q}, \ldots$ denote points on elliptic curves.

3 Related Work and Outline of the Idea

Lightweight applications are typically asymmetric in the sense that one of the communicating parties is strictly limited in resources, whereas the other is not. As an example, we consider an application where a wireless tag communicates with a server over a radio channel. The tag is limited in computational resources, power, and energy but the server has plenty of resources for computations. The tag implements an elliptic curve cryptosystem which requires both elliptic curve operations and modular arithmetic with integers (e.g., it signs messages with ECDSA [21]). The tag uses Koblitz curves for efficient scalar multiplication resulting in a need for obtaining both τ-adic expansions and their integer equivalents. In the following, we survey two existing options for implementing the above scheme as well as a new idea which allows delegating the expensive conversions from the tag to the powerful server.

3.1 Survey of the Existing Options

The first option, which is depicted in Figure 1(a), is to generate k as a random integer and convert it into a τ-adic expansion K for scalar multiplication.

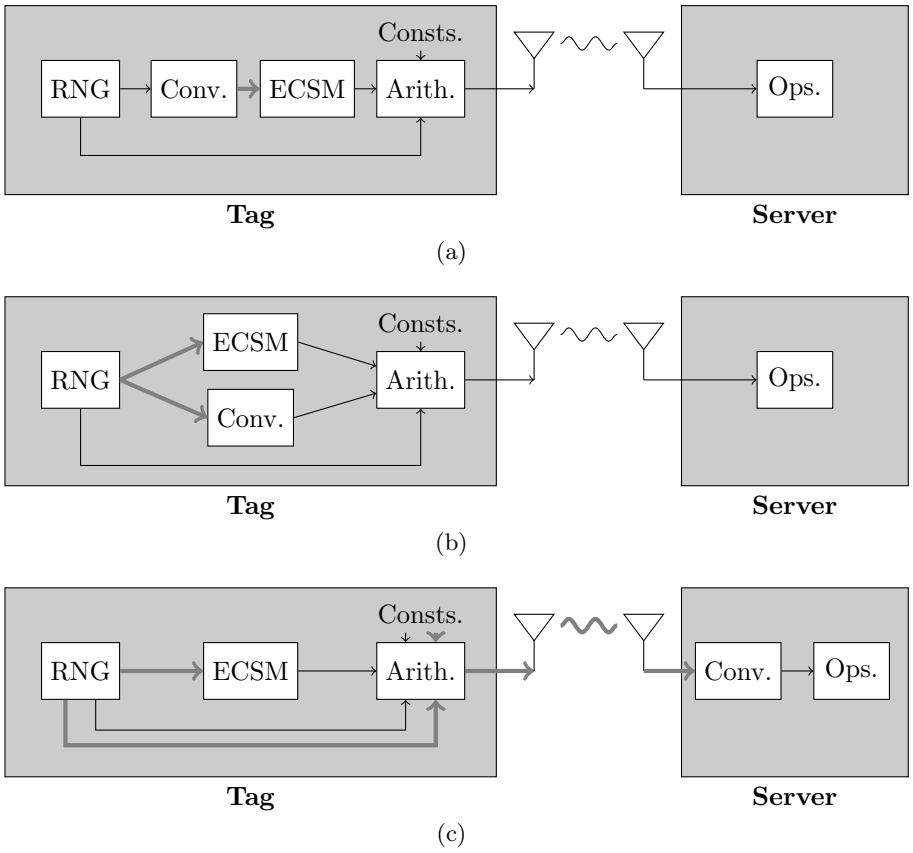

Fig. 1. Three options for using Koblitz curves on a wireless tag. Thin black arrows and thick gray arrows represent integer and τ-adic values, respectively. (a) the random number generator (RNG) generates scalar k as an integer which is converted to a τ-adic expansion K in order to use it in the elliptic curve scalar multiplication (ECSM) but k can be used as it is in the arithmetic part; (b) the RNG generates a random τ-adic expansion K which is used as it is in the ECSM but it is converted into an integer k in order to use it in the arithmetic part; and (c) the RNG generates a random τ-adic expansion K but the arithmetic part is also performed (at least partly) in the τ-adic domain. The computationally expensive conversion is delegated to the server.

The arithmetic part can be computed using the original integer k. The first method for conversion was given by Koblitz [13]. It has the drawback that the length of the τ-adic expansion is twice the length of the original scalar, consequently, reducing the efficiency of the scalar multiplication. Later, Meier and Staffelbach [17] and Solinas [24] showed that expansions of approximately the same length as the original scalar can be found. Solinas [24] also showed how to find τ-adic nonadjacent form (τNAF) and windowed NAF (w-τNAF) representations for the scalar k. These algorithms require, e.g., operations with large

rational numbers, which render them very inefficient for hardware implementations. The first hardware oriented conversion algorithm and implementation was presented by Järvinen et al. [10]. Brumley and Järvinen [6] later presented an algorithm requiring only integer additions, which resulted in the most compact hardware converter to date; however, even it is too large for very constrained devices mostly because it uses long adders and a high number of registers. Their work was extended by Adikari et al. [1] and Sinha Roy et al. [23] who focused on improving speed at the expense of resource requirements, which makes them even less suitable for constrained devices.

The second option, which is shown in Figure 1(b), is to generate the scalar as a random τ-adic expansion K and to find its integer equivalent for the arithmetic part. Generating random τ-adic expansions was first mentioned (and credited to Hendrik Lenstra) by Koblitz [13] but he did not provide a method for finding the integer equivalent of the scalar. The first method for retrieving the integer equivalent k was proposed by Lange in [15]. Her method requires several multiplications with long operands. More efficient methods were later introduced by Brumley and Järvinen in [5, 6]. The resource requirements of their methods are smaller than computing conversions to the other direction [6] but even they are too expensive for lightweight implementations.

3.2 Outline of the New Idea

In this paper, we propose a third option which, to the best of our knowledge, is not previously available in the literature. This option is shown in Figure 1(c). Similarly to the second option, the tag generates a random τ-adic expansion K and uses it for scalar multiplication. However, the tag does not compute the integer equivalent k but, instead, it uses K directly and all operations which depend on it are computed in the τ-adic domain. The results of these operations (τ-adic expansions) are transmitted over the radio channel to the server, which first converts the results to integers and then proceeds with normal server-side operations. Only the operations which depend on the scalar need to be computed in the τ-adic domain and, hence, it may be possible to compute other operations (and transmit their results) using integers. Clearly, this option improves efficiency of the tag only if operations in the τ-adic domain are cheap. In the following, we show that they can, indeed, be implemented with very few resources. From security perspective, the third option is equivalent with the second option discussed in Section 3.1 (c.f. [15]) because transmitting τ-adic expansions instead of their integer equivalents does not reveal any additional information about the secret scalars.

The new idea has similarities with [4], which presented a modified version of the Girault-Poupard-Stern identification scheme that handles only τ-adic expansions. Both [4] and the new idea use arithmetic in the τ-adic domain. We adapt and further develop the addition algorithm from [4]. The new idea allows delegating conversions to the more powerful party for arbitrary cryptosystems that require scalar multiplications on Koblitz curves and modular integer arithmetic with the scalar, whereas [4] presented a single identification scheme built around

τ-adic expansions only. For instance, it is unclear how to build a digital signature scheme that uses only τ-adic expansions because the ideas of [4] cannot be directly generalized to other schemes. We also provide the first hardware realizations of algorithms required to implement the new idea. These implementations may have importance also for implementing the scheme from [4].

4 Addition in the τ-adic Domain

The cornerstone of the idea discussed in Section 3.2 is to devise an efficient algorithm for adding two τ-adic expansions. In this section, we show how to construct such an algorithm. Our addition algorithm is close to the algorithm from [4] but we improve its efficiency and provide an analysis of the algorithm. Other arithmetic operations can be built upon the addition algorithm and they are discussed later in Section 5.

Let A and B be the τ-adic expansions of two positive integers a and b such that

$$A = \sum_{i=0}^{n-1} A_i \tau^i \quad \text{and} \quad B = \sum_{i=0}^{n-1} B_i \tau^i \tag{3}$$

where $A_i \in \{0, 1\}$ and $B_i \in \{-1, 0, 1\}$ so that $A_{n-1} = 1$ and/or $B_{n-1} = \pm 1$. Signed bits are allowed for B for two reasons: (a) Koblitz curve cryptosystems are typically implemented by using the τNAF) representation [24] or some other representation with signed bits (e.g., [22, 27]) and (b) this allows computing subtractions with the same algorithm.

Adding the two expansions gives the following expansion:

$$C = A + B = \sum_{i=0}^{n-1} C_i \tau^i \tag{4}$$

where $C_i = A_i + B_i \in \{-1, 0, 1, 2\}$. This expansion is correct in the sense that $C \stackrel{\circ}{=} a + b$ but it has several drawbacks because the set of digits has grown. Hence, the expansion must be processed in order to obtain a binary-valued τ-adic expansion. Instead of allowing C to have signed binary values as in [4], we limit the set of digits to unsigned binary values (i.e., $C_i \in \{0, 1\}$) in order to decrease the storage requirements for C. This does not imply restrictions for the use of the addition algorithm in our case as long as B_i are allowed to have signed binary values because we do not use the results of additions for computing scalar multiplications.

The binary-valued expansion C can be found analogously to normal addition of binary numbers by using a carry [4]. The main difference is that the carry is a τ-adic number t. The unsigned binary valued C_i is obtained by adding the coefficients A_i and B_i with the carry from the previous iteration and by reducing this value modulo 2; i.e., by taking the least significant bit (lsb). Every τ-adic number can be represented as $t_0 + t_1\tau$ where $t_0, t_1 \in \mathbb{Z}$ [24] and, hence, also the carry t can be represented with two integer coefficients as $t = t_0 + t_1\tau$.

Input: τ-adic expansions $A = \sum_{i=0}^{n-1} A_i \tau^i \stackrel{\circ}{=} a$ and $B = \sum_{i=0}^{n-1} B_i \tau^i \stackrel{\circ}{=} b$
Output: $C = \sum_{i=0}^{n'-1} C_i \tau^i$, where $C_i \in \{0, 1\}$, such that $C \stackrel{\circ}{=} a + b$

1 $(t_0, t_1) \leftarrow (0, 0)$; $i \leftarrow 0$
2 **while** $i < n$ **or** $(t_0, t_1) \neq (0, 0)$ **do**
3 $\quad r \leftarrow A_i + B_i + t_0$
4 $\quad C_i \leftarrow r \bmod 2$
5 $\quad (t_0, t_1) \leftarrow (t_1 + \mu \lfloor r/2 \rfloor, -\lfloor r/2 \rfloor)$
6 $\quad i \leftarrow i + 1$
7 **return** C

Algorithm 1: Addition in the τ-adic domain

Updating the carry for the next iteration requires a division by τ. As shown by Solinas [24], $t_0 + t_1\tau$ is divisible by τ if and only if t_0 is even. Subtracting C_i (rounding towards the nearest smaller integer) ensures this and, hence, we get:

$$((t_0 - C_i) + t_1\tau)/\tau = t_1 + \mu \left\lfloor \frac{t_0}{2} \right\rfloor - \left\lfloor \frac{t_0}{2} \right\rfloor \tau \ . \tag{5}$$

We continue the above process for all n bits of the operands and as long as $(t_0, t_1) \neq (0, 0)$. The resulting algorithm is shown in Algorithm 1.

Remark 1. Computing subtractions with Algorithm 1 is straightforward: $A - B = A + (-B) = A + \sum_{i=0}^{n-1} (-B_i)\tau^i$. I.e., we flip the signs of B_i and compute an addition with Algorithm 1. Alternatively, we revise Algorithm 1 so that Line 3 is replaced with $r \leftarrow A_i - B_i + t_0$.

4.1 Analysis of Algorithm 1

There are certain aspects that must be analyzed before Algorithm 1 is ready for efficient hardware implementation. The most crucial one is the sizes of the carry values t_0 and t_1 because efficient hardware implementation is impossible without knowing the number of flip-flops required for the carry. The ending condition of Algorithm 1 also implies that the latency of an addition depends on the values of the operands. This might open vulnerabilities against timing attacks. The following analysis sheds light on these aspects and provides efficient solutions for them.

In order to analyze Algorithm 1, we model it as a finite state machine (FSM) so that the carry (t_0, t_1) represents the state. Algorithm 1 can find unsigned binary τ-adic expansions with any $A_i, B_i \in \mathbb{Z}$ but, in this analysis and in the following propositions, we limit them so that $A_i \in \{0, 1\}$ and $B_i \in \{-1, 0, 1\}$, as described above. The FSM is constructed starting from the state $(t_0, t_1) = (0, 0)$ by analyzing all transitions with all possible inputs $A_i + B_i \in \{-1, 0, 1, 2\}$. E.g., when $\mu = 1$, we find out that the possible next states from the initial state $(0, 0)$ are $(0, 0)$ with inputs 0 and 1 (the corresponding outputs are then 0 and 1), $(-1, 1)$ with input -1 (output 1), and $(1, -1)$ with input 2 (output 0). Next, we

analyze $(-1, 1)$ or $(1, -1)$, and so on. The process is continued as long as there are states that have not been analyzed. The resulting FSM for $\mu = 1$ is depicted in Figure 2 and it contains 21 states. We draw two major conclusions from this FSM (and the corresponding one for $\mu = -1$).

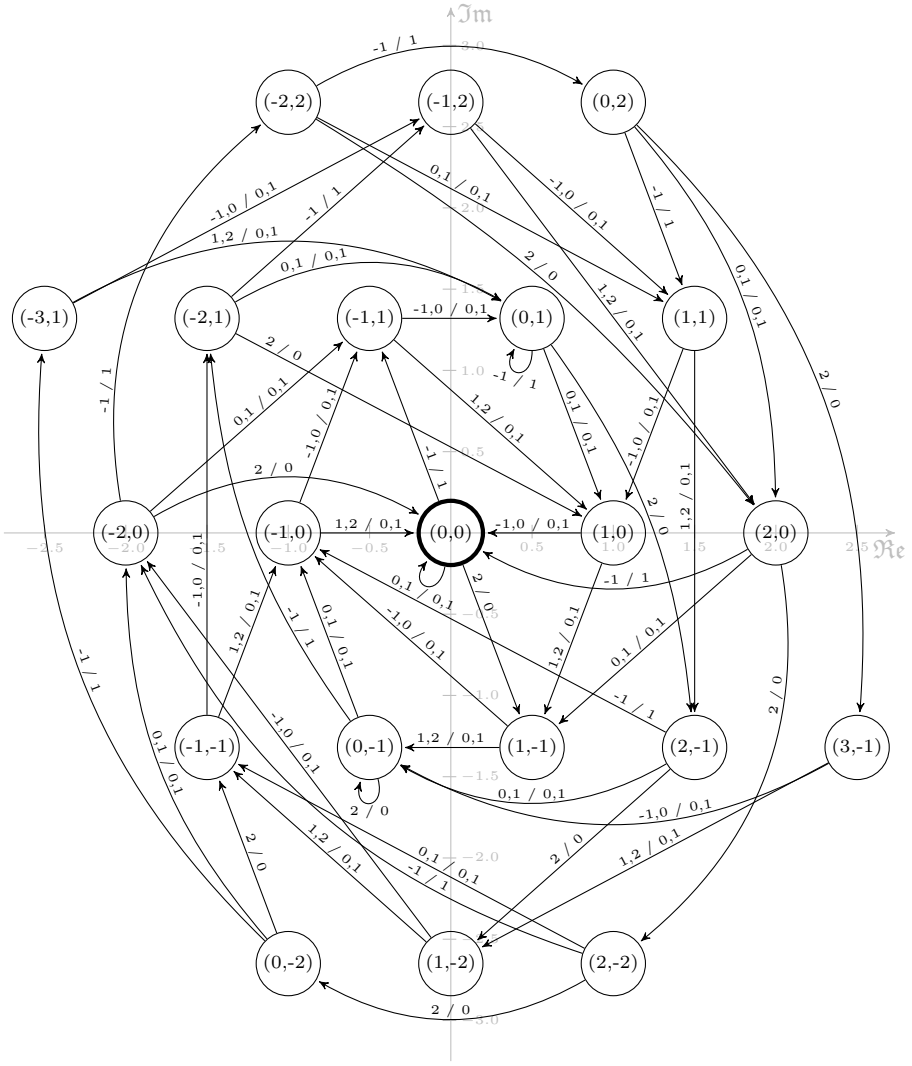

Fig. 2. The FSM for Algorithm 1, when $\mu = 1$, with inputs $A_i \in \{0, 1\}$ and $B_i \in \{-1, 0, 1\}$. The FSM is plotted on the complex plane so that each state is positioned based on its complex value $t = t_0 + t_1\tau$. The states are labeled with (t_0, t_1). State transitions are marked with *in* / *out* where *in* are the input values for which the transition is taken and *out* are the corresponding outputs.

Proposition 1. *For both $\mu = \pm 1$, the carry (t_0, t_1) of Algorithm 1 can be represented with 6 bits so that both t_0 and t_1 require 3 bits.*

Proof. The FSM of Figure 2 directly shows that $-3 \leq t_0 \leq 3$ and $-2 \leq t_1 \leq 2$. There are 7 distinct values for t_0 and 5 for t_1 and, hence, they can be represented with 3 bits, e.g., by using two's complement representation. The FSM for $\mu = -1$ can be constructed similarly and it also contains 21 states so that $-3 \leq t_0 \leq 3$ and $-2 \leq t_1 \leq 2$. Hence, t_0 and t_1 both require 3 bits for $\mu = \pm 1$. Consequently, the carry requires 6 bits. □

Remark 2. The FSM of Figure 2 includes 21 states. Hence, the states could be represented with only 5 bits. Unfortunately, if the algorithm is implemented directly as an FSM, the growth in the size of the combinational part outweighs the benefits gained from the lower number of flip-flops.

Proposition 2. *Let n be the larger of the lengths of A and B; i.e., $A_{n-1} = 1$ and/or $B_{n-1} = \pm 1$. Then, Algorithm 1 returns C with a length n' that satisfies*

$$n' \leq n + \lambda \tag{6}$$

where $\lambda = 7$ for both $\mu = \pm 1$.

Proof. After all n bits of A and B have been processed, the FSM can be in any of the 21 states. Hence, the constant λ is given by the longest path from any state to the state $(0, 0)$ when the input is fixed to zero; i.e., $A_i = B_i = 0$. The FSM of Figure 2 shows that the longest path starts from the state $(0, 2)$ and goes through the following states $(2, 0)$, $(1, -1)$, $(0, -1)$, $(-1, 0)$, $(-1, 1)$, and $(0, 1)$ to $(0, 0)$ and outputs $(0, 0, 1, 1, 1, 0, 1)$. Thus, $\lambda = 7$ for $\mu = 1$. It can be shown similarly that $\lambda = 7$ also for $\mu = -1$. □

5 Other Operations in the τ-adic Domain

In this section, we describe algorithms for other arithmetic operations in the τ-adic domain, which are required in order to implement the idea of Section 3.2. The algorithms are based on using the addition algorithm given in Algorithm 1.

5.1 Folding

The length of an arbitrarily long τ-adic expansion can be reduced to about m bits without changing its integer equivalent modulo q, where q is the order of the base point of the scalar multiplication. The integer equivalent of a τ-adic expansion $A = \sum_{i=0}^{n-1} A_i \tau^i$ can be retrieved by computing the sum $a = \sum_{i=0}^{n-1} A_i s^i \pmod{q}$ where s, the integer equivalent of τ, is a per-curve constant integer [15]. Because $s^m \equiv 1 \pmod{q}$,

$$a = \sum_{i=0}^{n-1} A_i s^i \equiv \sum_{j=0}^{\lfloor n/m \rfloor} \sum_{i=0}^{m-1} A_{jm+i} s^i \pmod{q}, \tag{7}$$

Input: τ-adic expansion $A = \sum_{i=0}^{n-1} A_i \tau^i \stackrel{\circ}{=} a$, m, and $\ell \geq m$
Output: $B = \sum_{i=0}^{n'-1} B_i \tau^i \stackrel{\circ}{=} b = a$ and $n' \leq \ell$

$B \leftarrow A^{(0)}$
for $j = 1$ **to** $\lfloor n/m \rfloor$ **do**
  $B \leftarrow B + A^{(j)}$ `/* Algorithm 1 */`
while $n' > \ell$ **do**
  $B \leftarrow B^{(0)} + B^{(1)} + \ldots + B^{(\lfloor n'/m \rfloor)}$ `/* Optional, Algorithm 1 */`
return B

Algorithm 2: Folding

where $A_i = 0$ for $i \geq n$. As a result of (7), an expansion can be compressed to approximately m bits by "folding" the expansion; i.e., folding is analogous to modular reduction. Let $A^{(j)} = \sum_{i=0}^{m-1} A_{jm+i} \tau^i$, the j-th m-bit block of A. Then, an approximately m-bit τ-adic expansion B having the same integer equivalent with A can be obtained by computing $B = A^{(0)} + A^{(1)} + \ldots + A^{(\lfloor n/m \rfloor)}$ with $\lfloor n/m \rfloor$ applications of Algorithm 1. Because of the carry structure of Algorithm 1, the length of the expansion may still exceed m bits. Additional foldings can be computed in the end in order to trim the length of B below a predefined bound $\ell \geq m$. An algorithm for folding (including the optional trimming in the end) is given in Algorithm 2. In most practical cases, the optional trimming requires at most one addition: $B^{(0)} + B^{(1)}$.

5.2 Multiplication

Multiplication of two τ-adic expansions A and B is given as follows:

$$C = A \times B = \sum_{i=0}^{n-1} A_i \tau^i B \ . \tag{8}$$

An algorithm for multiplication in the τ-adic domain can be devised by using a variation of the binary method. An addition is computed with Algorithm 1 if $A_i = 1$ and a multiplication by τ is performed for all A_i by shifting the bit vector. Hence, multiplication requires $n - 1$ shifts and $\rho(A) - 1$ additions, where $\rho(A)$ is the Hamming weight of A. A bit-serial most significant bit (msb) first multiplication algorithm is presented in Algorithm 3. A similar multiplication algorithm was used also in [4].

It is also possible to use the binary method for computing multiplications where the other operand, say a, is an integer. Algorithm 4 presents a bit-serial msb first algorithm for computing $C = a \times B$ such that $C \stackrel{\circ}{=} a \times b$. It requires $n + \rho(A) - 2$ additions with Algorithm 1.

Remark 3. Algorithm 4 also serves as an algorithm for converting integers to the τ-adic domain. An integer a can be converted by computing $a \times 1$ with Algorithm 4. The algorithm returns $C = A$, the unsigned binary τ-adic expansion

Input: τ-adic expansions $A = \tau^{n-1} + \sum_{i=0}^{n-2} A_i \tau^i \triangleq a$, where $A_i \in \{0, 1\}$, and $B \triangleq b$, where $B_i \in \{-1, 0, 1\}$
Output: $C = A \times B$ such that $C \triangleq a \times b$

```
1 C ← B
2 for i = n − 2 to 0 do
3 |   C ← τC                                          /* Shift */
4 |   if A_i = 1 then
5 |   |   C ← C + B                                   /* Algorithm 1 */
6 return C
```

Algorithm 3: Multiplication in the τ-adic domain

Input: Integer $a = 2^{\lfloor \log_2 a \rfloor} + \sum_{i=0}^{\lfloor \log_2 a \rfloor - 1} a_i 2^i$, where $a_i \in \{0, 1\}$, and a τ-adic expansion $B \triangleq b$, where $B_i \in \{-1, 0, 1\}$
Output: C such that $C \triangleq a \times b$

```
1 C ← B
2 for i = ⌊log_2 a⌋ − 1 to 0 do
3 |   C ← C + C                                       /* Algorithm 1 */
4 |   if a_i = 1 then
5 |   |   C ← C + B                                   /* Algorithm 1 */
6 return C
```

Algorithm 4: Multiplication by an integer in the τ-adic domain

of a. This could also be used for converting k but, in that case, K would have $\rho(K) \approx n/2$, whereas representing K in τNAF gives $\rho(K) \approx n/3$ and results in more efficient scalar multiplications.

Remark 4. Different versions of the binary method can be straightforwardly used for devising an algorithm for multiplications of τ-adic expansions (also when the other operand is an integer). Especially, using Montgomery's ladder [19] would give an algorithm with a constant sequence of operations (shifts and additions), which would provide resistance against side-channel analysis. The scalar k is typically a nonce and the adversary is limited to a single side-channel trace. Thus, constant pattern of operations offers sufficient protection against most attacks.

5.3 Multiplicative Inverse

The multiplicative inverse modulo q, a^{-1}, for an integer a can be found via Fermat's Little Theorem by computing the following exponentiation:

$$a^{-1} = a^{q-2} \pmod{q} \ . \tag{9}$$

This exponentiation gives a straightforward way to compute inversions also with τ-adic expansions. Let $q' = q-2$. Given a τ-adic expansion A, a τ-adic expansion

Input: τ-adic expansion A of integer a and $q' = q - 2$
Output: B such that $b \equiv a^{-1} \pmod{q}$

```
1 B ← A
2 for i = ⌊log_2 q'⌋ − 1 to 0 do
3 |   B ← B × B                                   /* Algorithm 3 */
4 |   if q'_i = 1 then
5 |   |   B ← B × A                               /* Algorithm 3 */
6 return B
```

Algorithm 5: Inversion modulo q in the τ-adic domain

A^{-1} such that $A \times A^{-1} \stackrel{\circ}{=} a \times a^{-1} \equiv 1 \pmod{q}$ can be found by computing:

$$A^{-1} = A^{q'} = \prod_{i=0}^{\lfloor \log_2 q' \rfloor} A^{q'_i 2^i} \ . \tag{10}$$

Algorithm 5 shows an algorithm for computing (10) by using Algorithm 3.

6 Architecture

The objective of this work was to provide an efficient circuitry with small resource requirements that could be used for computing τ-adic arithmetic in lightweight implementations. Figure 3 presents an architecture that implements Algorithm 1 for $\mu = 1$. Because $B_i \in \{-1, 0, 1\}$, it can be used for K given using signed-bit representations (e.g., [22, 24, 27]). Because $t_0 \in [-3, 3]$ and $A_i + B_i \in [-1, 2]$, $r \in [-4, 5]$ and we need 4 bits to represent it. The division $\lfloor r/2 \rfloor$ can be trivially performed by dropping off the lsb of r, which is used directly as C_i. The carry values t_0 and t_1 are represented as 3-bit two's complement numbers (see Proposition 1). Hence, $-\lfloor r/2 \rfloor$ is obtained by flipping the bits and adding one, which results in the circuitry shown on the right in Figure 3. The while loop can be implemented as a for loop from 0 to $n + \lambda - 1$ (see Proposition 2). The rest of Algorithm 1 and the other algorithms (e.g., Algorithm 3) can be implemented with a simple control logic and shift registers for the operands. Algorithm 1 and the architecture of Figure 3 are independent of the field size m and, hence, the same architecture can be used for all Koblitz curves with $\mu = 1$.

A circuitry for $\mu = -1$ can be devised similarly but we omit the description because of space restrictions. We merely state that it is almost similar: the only difference is that the adders updating t_0 (on the left in Figure 3) use the outputs of the negation part that computes $-\lfloor r/2 \rfloor$ (on the right in Figure 3) instead of taking $\lfloor r/2 \rfloor$ directly. Hence, the area requirements should, in theory, remain the same but the critical path becomes longer by one NOT and two XORs (in the half adders).

The circuitry of Figure 3 computes additions in the τ-adic domain with a constant latency of $n + \lambda$ clock cycles. Assuming that $n \approx m$, we get that an

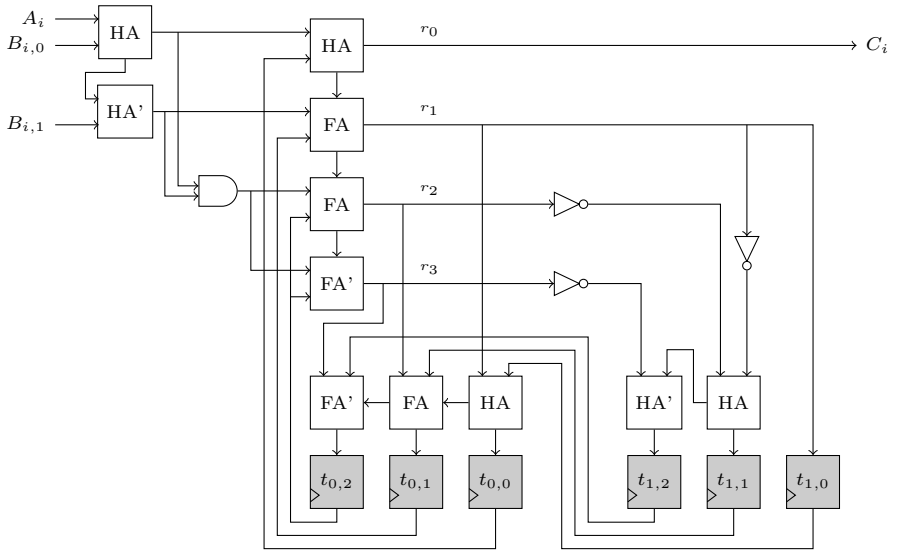

Fig. 3. Architecture for $\mu = 1$. The circuit consists of 4 half adders (HA), 3 full adders (FA), 2 half adders without carry logic (HA'), 2 full adders without carry logic (FA'), 3 NOTs, and 6 flip-flops. All wires are single bit wires.

addition requires $m+7$ clock cycles; this gives 170 clock cycles for the NIST curve K-163 from [21]. If multiplication by τ (shift) takes one clock cycle, Algorithm 3 requires approximately $m(m + \lambda + 2)/2$ clock cycles and Algorithm 4 requires approximately $3m(m+\lambda)/2$ clock cycles; this gives roughly 14000 or 41600 clock cycles, respectively, for NIST K-163. These latencies are small compared to the latency of scalar multiplication [2, 3, 8, 14, 16]. It is also typical for lightweight implementations that area, power, and energy consumption are more important than latency.

7 Results and Comparison

We described the circuitry of Figure 3 and the corresponding one for $\mu = -1$ in VHDL and simulated them with ModelSim SE 6.6d. We used Synopsys Design Compiler D-2010.03-SP4 with Faraday FSC0L standard cell libraries for synthesizing them for UMC 0.13 μm CMOS with voltage of 1.2 V. When synthesized using the 'compile ultra' process without additional constraints, the areas of the circuitries were 75.25 and 76.25 gate equivalents (GE) for $\mu = 1$ and $\mu = -1$, respectively.

The converter architectures available in the literature have been implemented on field-programmable gate arrays (FPGA) and, consequently, their area and performance characteristics are available only for FPGAs. Hence, comparing the circuitries presented above to the state-of-the-art converters is not straightforward. In order to perform a fair comparison, we estimate the GE counts of

Table 1. Comparison to the state-of-the-art converters for NIST K-163 ($\mu = 1$)

Work	Technology	Area / Notes	GE
[6], integer-to-τNAF	FPGA, Stratix II S60C4	948 ALUTs, 683 FFs	>7200
[6], τ-adic-to-integer	FPGA, Stratix II S60C4	850 ALUTs, 491 FFs	>3600
This work, $\mu = 1$	ASIC, 0.13 μm CMOS	Fig. 3	75.25
This work, $\mu = 1$	ASIC, 0.13 μm CMOS	Fig. 3, 340 FFs	∼2000

the converters from [6], which are the most compact converters available in the literature. These estimates in the case of NIST K-163 are collected in Table 1.

The integer to τNAF converter [6] includes two m-bit and four $m/2$-bit adders and registers as well as several multiplexers and comparators. A full adder and a flip-flop both require 5.5 GE on 0.13 μm CMOS and, hence, we can estimate that only the adders and registers occupy an area of about 7200 GE if $m = 163$. The area of the τ-adic expansion to integer converter [6] that requires two m-bit adders, two m-bit registers, multiplexers, and comparators can be estimated similarly. The adders and registers alone give an area estimate of about 3600 GE if $m = 163$.

Algorithms 3–4 require two $(m + \lambda)$-bit registers. We anticipate that in most implementations these registers can be shared with the circuitry computing scalar multiplications. In that case, the overhead of the circuitries is only about one hundred GEs (including the control logic), which is negligible compared to the converters. If none of these registers can be shared with the scalar multiplier, then the circuitry for NIST K-163 including registers has an area of approximately 2000 GE. This area is still only about half of the area of the smallest converter available today.

8 Case Study: ECDSA

In this section, we present a case study of how the new scheme could be used for ECDSA. The tag computes an ECDSA signature for a message $\mathcal{M}$ and sends it to a more powerful server for verification. The signature (r, s) is computed as follows [21]:

$$k \in_R [1, q-1] \tag{11}$$

$$r = [k\mathbf{P}]_x \tag{12}$$

$$e = H(\mathcal{M}) \tag{13}$$

$$s = k^{-1}(e + dr) \bmod q \tag{14}$$

where d is the signer's private key, $[k\mathbf{P}]_x$ denotes the x-coordinate of the result point of the scalar multiplication $k\mathbf{P}$, and $H(\mathcal{M})$ is the hash value of $\mathcal{M}$ (e.g., SHA-256).

Equation (12) can be efficiently computed using Koblitz curves if k is given as a τ-adic expansion; i.e., we compute $r = [K\mathbf{P}]_x$. We can use the τNAF

representation for K in order to speedup computations. If the compact encoding proposed by Joye and Tymen [11] is used, then K can be obtained by generating m random bits. In order to avoid computing the expensive inversion of (14), we can transmit the nominator and denominator separately after blinding them with $b \in_R [1, q-1]$ as proposed in [20]: $s_n = b(e+dr) \bmod q$ and $s_d = bk \bmod q$. Because K affects only s_d, we compute s_n using cheaper integer arithmetic. The denominator can be computed with a single multiplication in the τ-adic domain: $S_d = b \times K$ by using Algorithm 4. The result of the multiplication should be compressed by folding it with Algorithm 2 after (and at any time during) the execution of Algorithm 4. Instead of transmitting a $2m$-bit (r, s), we now transmit approximately $3m$-bit (r, s_n, S_d). The server computes s_d from S_d and performs the modular division $s = s_n/s_d \pmod q$, after which it proceeds normally with the signature verification procedure from [21].

If transmission is expensive, the transmittable amount can be reduced to $2m$ bits by computing the inversion in the tag and transmitting (r, S). In this case, it is preferable to compute $e+dr$ using integers, invert K using Algorithm 5, and compute $S = (e+dr) \times K^{-1}$ with Algorithm 4. Both S and intermediate values should be folded with Algorithm 2 in order to limit the amount of storage and transmission. In this case, the server simply converts S to s before proceeding normally.

9 Conclusions and Future Work

In this paper, we showed that, contrary to previous beliefs, Koblitz curves can be efficiently used in lightweight implementations even if integer arithmetic is required with the scalar k. Because Koblitz curves offer more efficient scalar multiplications compared to general binary curves, utilizing the findings of this paper will probably enable more efficient lightweight implementations of ECC than what has been possible in the past. We conclude with the following suggestions for future research:

Future work 1. For Koblitz curve cryptosystems, resistance against side-channel attacks can be achieved by using dummy point additions [7], randomized representations for the scalar [7], or more efficiently with a zerofree representation for the scalar [22, 27]. The approach presented in this paper can be straightforwardly applied also in these cases. As mentioned in Remark 4, the side-channel resistivity of the algorithms proposed in this paper can be improved, e.g., by using Montgomery's ladder [19] in Algorithms 3 and 4. The circuitries of Section 6 can be implemented with secure logic styles (e.g., [26]) in order to limit side-channel leakage. Although the more significant side-channel leakages are typically in scalar multiplication parts, resistance against side-channel attacks deserves further research in the future.

Future work 2. The registers for t occupy almost half of the areas of the addition circuits. Hence, significant speedups and area-speed ratio improvements could be achieved by processing several A_i and B_i in one iteration because this would affect only the amount of logic, not the number of flip-flops.

Future work 3. As discussed in Section 7, the circuitries have negligible area overheads if the shift registers for the operands can be shared with the circuitry computing scalar multiplications. It will be studied in the future how registers could be shared, e.g., with the compact architecture presented in [2].

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments and improvement suggestions. The work was partly funded by KU Leuven under GOA TENSE (GOA/11/007) and the F+ fellowship (F+/13/039) and by the Hercules Foundation (AKUL/11/19).

References

1. Adikari, J., Dimitrov, V., Järvinen, K.: A fast hardware architecture for integer to τNAF conversion for Koblitz curves. IEEE Transactions on Computers 61(5), 732–737 (May 2012)
2. Azarderakhsh, R., Järvinen, K.U., Mozaffari-Kermani, M.: Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications. IEEE Transactions on Circuits and Systems I—Regular Papers 61(4), 1144–1155 (Apr 2014)
3. Batina, L., Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Low-cost elliptic curve cryptography for wireless sensor networks. In: Proc. the 3rd European Workhop on Security and Privacy in Ad-Hoc and Sensor Networks — ESAS 2006. Lecture Notes in Computer Science, vol. 4357, pp. 6–17. Springer (2006)
4. Benits, Jr., W.D., Galbraith, S.D.: The GPS identification scheme using Frobenius expansions. In: Western European Workshop on Research in Cryptology — WEWoRC 2007. Lecture Notes in Computer Science, vol. 4945, pp. 13–27. Springer (2008)
5. Brumley, B.B., Järvinen, K.: Koblitz curves and integer equivalents of Frobenius expansions. In: Selected Areas in Cryptography — SAC 2007. Lecture Notes in Computer Science, vol. 4876, pp. 126–137. Springer (2007)
6. Brumley, B.B., Järvinen, K.U.: Conversion algorithms and implementations for Koblitz curve cryptography. IEEE Transactions on Computers 59(1), 81–92 (Jan 2010)
7. Hasan, M.A.: Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems. IEEE Transactions on Computers 50(10), 1071–1083 (Oct 2001)
8. Hein, D.M., Wolkerstorfer, J., Felber, N.: ECC is ready for RFID - a proof in silicon. In: Selected Areas in Cryptography — SAC 2008. Lecture Notes in Computer Science, vol. 5381, pp. 401–413. Springer (2009)
9. Järvinen, K.: Optimized FPGA-based elliptic curve cryptography processor for high-speed applications. Integration, the VLSI Journal 44(4), 270–279 (2011)
10. Järvinen, K., Forsten, J., Skyttä, J.: Efficient circuitry for computing τ-adic non-adjacent form. In: Proc. the 13th IEEE International Conference on Electronics, Circuits and Systems — ICECS 2006. pp. 232–235. IEEE (2006)
11. Joye, M., Tymen, C.: Compact encoding of non-adjacent forms with applications to elliptic curve cryptography. In: Public Key Cryptography — PKC 2001. Lecture Notes in Computer Science, vol. 1992, pp. 353–364. Springer (2001)

12. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48(177), 203–209 (1987)
13. Koblitz, N.: CM-curves with good cryptographic properties. In: Advances in Cryptology — CRYPTO '91. Lecture Notes in Computer Science, vol. 576, pp. 279–287. Springer (1991)
14. Koçabas, Ü., Fan, J., Verbauwhede, I.: Implementation of binary Edwards curves for very-constrained devices. In: Proc. the 21st IEEE International Conference on Application-specific Systems Architectures and Processors — ASAP 2010. pp. 185–191. IEEE (2010)
15. Lange, T.: Koblitz curve cryptosystems. Finite Fields and Their Applications 11, 200–229 (2005)
16. Lee, Y.K., Sakiyama, K., Batina, L., Verbauwhede, I.: Elliptic-curve-based security processor for RFID. IEEE Transactions on Computers 57(11), 1514–1527 (2008)
17. Meier, W., Staffelbach, O.: Efficient multiplication on certain nonsupersingular elliptic curves. In: Advances in Cryptology — CRYPTO '92. Lecture Notes in Computer Science, vol. 740, pp. 333–344. Springer (1993)
18. Miller, V.S.: Use of elliptic curves in cryptography. In: Advances in Cryptology — CRYPTO '85. Lecture Notes in Computer Science, vol. 218, pp. 417–426. Springer (1986)
19. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48, 243–264 (1987)
20. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? Complexity trade-offs with the digital signature algorithm. In: Advances in Cryptology — EUROCRYPT '94. Lecture Notes in Computer Science, vol. 950, pp. 77–85. Springer (1994)
21. National Institute of Standards and Technology (NIST): Digital signature standard (DSS). Federal Information Processing Standard, FIPS PUB 186-4 (Jul 2013)
22. Okeya, K., Takagi, T., Vuillaume, C.: Efficient representations on Koblitz curves with resistance to side channel attacks. In: Proc. the 10th Australasian Conference on Information Security and Privacy — ACISP 2005. Lecture Notes in Computer Science, vol. 3574, pp. 218–229. Springer (2005)
23. Sinha Roy, S., Fan, J., Verbauwhede, I.: Accelerating scalar conversion for Koblitz curve cryptoprocessors on hardware platforms. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (to appear)
24. Solinas, J.A.: Efficient arithmetic on Koblitz curves. Designs, Codes and Cryptography 19(2–3), 195–249 (2000)
25. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. Journal of Cryptographic Engineering 1(3), 187–199 (2011)
26. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: Proc. Design, Automation and Test in Europe Conference and Exhibition — DATE 2004. vol. 1, pp. 246–251. IEEE (2004)
27. Vuillaume, C., Okeya, K., Takagi, T.: Defeating simple power analysis on Koblitz curves. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A(5), 1362–1369 (May 2006)