Koblitz Curves and Integer Equivalents of Frobenius Expansions*

Billy Bob Brumley[1] and Kimmo Järvinen[2]

[1] Laboratory for Theoretical Computer Science, Helsinki University of Technology,
P.O. Box 5400, FIN-02015 TKK, Finland
billy.brumley@tkk.fi

[2] Signal Processing Laboratory, Helsinki University of Technology,
P.O. Box 3000, FIN-02015 TKK, Finland
kimmo.jarvinen@tkk.fi

Abstract. Scalar multiplication on Koblitz curves can be very efficient due to the elimination of point doublings. Modular reduction of scalars is commonly performed to reduce the length of expansions, and τ-adic Non-Adjacent Form (NAF) can be used to reduce the density. However, such modular reduction can be costly. An alternative to this approach is to use a random τ-adic NAF, but some cryptosystems (e.g. ECDSA) require both the integer and the scalar multiple. This paper presents an efficient method for computing integer equivalents of random τ-adic expansions. The hardware implications are explored, and an efficient hardware implementation is presented. The results suggest significant computational efficiency gains over previously documented methods.

Keywords: Koblitz curves, elliptic curve cryptography, digital signatures.

1 Introduction

While compact keys and signatures occur naturally when using elliptic curves, the computational efficiency of elliptic curve cryptosystems is the subject of much research. Koblitz [1] showed that scalar multiplication can be done very fast on a certain family of binary curves now commonly referred to as Koblitz curves. In the same paper, Koblitz credited Hendrik Lenstra for first suggesting random base-τ expansions for key agreement protocols using Koblitz curves.

Meier and Staffelbach [2] showed how to significantly reduce the length of τ-adic expansions by performing modular reduction on scalars. Solinas [3,4] later built on this idea and additionally reduced the weight by designing a τ-adic analogue of Non-Adjacent Form (NAF).

Unfortunately, performing such modular reduction can be costly. As future work, Solinas suggested a study of the distribution of random τ-adic NAFs. Lange and Shparlinski [5,6] have studied the distribution of such expansions in depth.

* This work was supported by the project "Packet Level Authentication" funded by TEKES.

For key agreement protocols like Diffie-Hellman [7], the integer equivalent of such a random τ-adic expansion is not needed. However, for ElGamal [8] type digital signatures like ECDSA [9], both the integer and the scalar multiple are needed to generate a signature. Lange [10] discussed many of the details of this approach, as well a straightforward method for recovering the integer equivalent using a number of multiplications.

In this paper, we present a new method for recovering integer equivalents of random τ-adic expansions using only additions and one field multiplication. This method is shown to be very efficient and has significant hardware implications. A hardware implementation is also presented and studied in depth. The results are then compared to current similar methods in hardware.

Sec. 2 reviews background information on Koblitz curves and τ-adic expansions. Sec. 3 covers more recent research on random τ-adic expansions, as well as our new method for efficiently computing integer equivalents of such expansions. Sec. 4 presents an efficient hardware implementation of our new method on a field programmable gate array (FPGA), as well as a comparison to current methods. We conclude in Sec. 5.

2 Koblitz Curves

Koblitz curves [1] are non-supersingular elliptic curves defined over $\mathbb{F}_2$, i.e.

$$E_a(\mathbb{F}_{2^m}) : y^2 + xy = x^3 + ax^2 + 1 \text{ where } a \in \{0, 1\} \ . \tag{1}$$

These curves have the nice property that if a point $P = (x, y)$ is on the curve, so is the point (x^2, y^2). The map $\sigma : E_a(F_{2^m}) \rightarrow E_a(F_{2^m});\ (x, y) \mapsto (x^2, y^2)$ is called the Frobenius endomorphism. From the point addition formula, it can be shown that for all $(x, y) \in E_a$

$$\begin{aligned}(x^4, y^4) + 2(x, y) = \mu(x^2, y^2), &\text{ where } \mu = (-1)^{1-a}, \text{ equivalently}\\ (\sigma^2 + 2)P = \mu\sigma P, &\text{ and hence}\\ \sigma^2 + 2 = \mu\sigma &\text{ as curve endomorphisms.}\end{aligned} \tag{2}$$

The last equation also allows us to look at σ as a complex number τ and we can extend scalar multiplication to scalars $d_0 + d_1\tau$ from $\mathbb{Z}[\tau]$. Higher powers of τ make sense as repeated applications of the endomorphisms.

Koblitz showed how to use use the Frobenius endomorphism very efficiently in scalar multiplication: A scalar $n = d_0 + d_1\tau$ is expanded using (2) repeatedly, i.e. put $\epsilon_0 = d_0 \pmod 2$ and replace n by $(d_0 - \epsilon_0)/2\mu + d_1 - (d_0 - \epsilon_0)/2\tau$ to compute ϵ_1 etc. This iteration leads to a so called τ-adic expansion with coefficients $\epsilon \in \{0, 1\}$ such that $nP = \sum_{i=0} \epsilon_i \tau^i(P)$.

As a result of representing integers as the sum of powers of τ, scalar multiplication can be accomplished with no point doublings by combining point additions and applications of σ. Koblitz noted in [1] that such base-τ expansions unfortunately have twice the length when compared to binary expansions. This leads to twice the number of point additions on average.

2.1 Integer Equivalents

To overcome this drawback, Meier and Staffelbach [2] made the following observation. Given any point $P \in E_a(\mathbb{F}_{2^m})$, it follows that

$$\begin{aligned} P &= (x, y) = (x^{2^m}, y^{2^m}) = \tau^m P \\ \mathcal{O} &= (\tau^m - 1)P. \end{aligned}$$

Two elements $\gamma, \rho \in \mathbb{Z}[\tau]$ such that $\gamma \equiv \rho \pmod{\tau^m - 1}$ are said to be *equivalent* with respect to P as γ multiples of P can also be obtained using the element ρ since for some κ

$$\gamma P = \rho P + \kappa(\tau^m - 1)P = \rho P + \kappa \mathcal{O} = \rho P \ .$$

While this relation holds for all points on the curve, cryptographic operations are often limited to the main subgroup; that is, the points of large prime order. Solinas [4] further improved on this. The small subgroup can be excluded since $(\tau - 1)$ divides $(\tau^m - 1)$, and for multiples of points in the main subgroup scalars can be reduced modulo $\delta = (\tau^m - 1)/(\tau - 1)$ to further reduce the length to a maximum of $m + a$. For computational reasons, it is often convenient to have the form $\delta = c_0 + c_1\tau$ as well. For reference, the procedure for performing such modular reduction is presented as Algorithm 1, which calculates ρ' such that the probability that $\rho' \neq \rho$ holds is less than $2^{-(C-5)}$. This probabilistic approach is used to avoid costly rational divisions, trading them for a number of multiplications.

Algorithm 1. Partial reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$.

```
Input: Integer $n$, constants $C > 5$, $s_0 = c_0 + \mu c_1$, $s_1 = -c_1$,
       $V_m = 2^m + 1 - \#E_a(\mathbb{F}_{2^m})$.
Output: $n$ partmod $\delta$.
$n' \leftarrow \lfloor n/2^{a-C+(m-9)/2} \rfloor$
for $i \leftarrow 0$ to $1$ do
    $g' \leftarrow s_i n'$
    $j' \leftarrow V_m \lfloor g'/2^m \rfloor$
    $\lambda_i \leftarrow \lfloor (g' + j')/2^{(m+5)/2} + \frac{1}{2} \rfloor / 2^C$
end
$(q_0, q_1) \leftarrow$ Round$(\lambda_0, \lambda_1)$                    /* Using Algorithm 2 */
$d_0 \leftarrow k - (s_0 + \mu s_1)q_0 - 2s_1q_1$, $d_1 \leftarrow s_1q_0 - s_0q_1$
return $d_0 + d_1\tau$
```

Solinas [4] also developed a τ-adic analogue of Non-Adjacent Form (NAF). Signed representations are used as point subtraction has roughly the same cost as point addition; NAF guarantees that no two adjacent coefficients are non-zero. By reducing elements of $\mathbb{Z}[\tau]$ modulo δ and using τ-adic NAF, such expansions have roughly the same length and average density ($1/3$ when $\epsilon_i \in \{0, 1, -1\}$) as normal NAFs of the same integers.

Algorithm 2. Rounding off in $\mathbb{Z}[\tau]$.

```
Input: Rational numbers $\lambda_0$ and $\lambda_1$.
Output: Integers $q_0, q_1$ such that $q_0 + q_1\tau$ is close to $\lambda_0 + \lambda_1\tau$.
for $i \leftarrow 0$ to $1$ do
    $f_i \leftarrow \lfloor \lambda_i + \frac{1}{2} \rfloor$
    $\eta_i \leftarrow \lambda_i - f_i$, $h_i \leftarrow 0$
end
$\eta \leftarrow 2\eta_0 + \mu\eta_1$
if $\eta \geq 1$ then
    if $\eta_0 - 3\mu\eta_1 < -1$ then $h_1 \leftarrow \mu$
    else $h_0 \leftarrow 1$
end
else
    if $\eta_0 + 4\mu\eta_1 \geq 2$ then $h_1 \leftarrow \mu$
if $\eta < -1$ then
    if $\eta_0 - 3\mu\eta_1 \geq 1$ then $h_1 \leftarrow -\mu$
    else $h_0 \leftarrow -1$
end
else
    if $\eta_0 + 4\mu\eta_1 < -2$ then $h_1 \leftarrow -\mu$
$q_0 \leftarrow f_0 + h_0$, $q_1 \leftarrow f_1 + h_1$
return $q_0, q_1$
```

3 Random τ-adic Expansions

Instead of performing such non-trivial modular reduction, Solinas [4] suggested (an adaptation of an idea credited to Hendrik Lenstra in Koblitz's paper [1]) producing a *random* τ-adic NAF; that is, to build an expansion by generating digits with $\Pr(-1) = 1/4$, $\Pr(1) = 1/4$, $\Pr(0) = 1/2$, and following each non-zero digit with a zero. Lange and Shparlinski [5] proved that such expansions with length $\ell = m - 1$ are well-distributed and virtually collision-free.

This gives an efficient way of obtaining random multiples of a point (for example, a generator). For some cryptosystems (e.g. Diffie-Hellman [7]), the integer equivalent of the random τ-adic NAF is not needed. The expansion is simply applied to the generator, then to the other party's public point. However, for generating digital signatures (e.g. ECDSA [9]), the equivalent integer is needed.

Lange [10] covered much of the theory of this approach, as well as a method for recovering the integer. Given a generator G of prime order r and a group automorphism σ, there is a unique integer s modulo r which satisfies $\sigma(G) = sG$, and s (fixed per-curve) is obtained using $(T - s) = \gcd((T^m - 1)/(T - 1), T^2 - \mu T + 2)$ in $\mathbb{F}_r[T]$. We note that s also satisfies $s = (-c_0)(c_1)^{-1} \pmod{r}$. Values of s for standard Koblitz curves are listed in Table 1 for convenience.

Given some τ-adic expansion $\sum_{i=0}^{\ell-1} \epsilon_i \tau^i$ with $\epsilon_i \in \{0, 1, -1\}$, it follows that the equivalent integer can be recovered deterministically as $\sum_{i=0}^{\ell-1} \epsilon_i s^i$ using at most $\ell - 2$ multiplications and some additions for non-zero coefficient values, all modulo r [10]. Our approach improves on these computation requirements

Table 1. Integer s such that $\sigma P = sP$ modulo r on $E_a(\mathbb{F}_{2^m})$

Curve	a	s					
K-163	1	00000003	81afd9e3	493dccbf	c2faf1d2	84e6d34e	bd67a6da
K-233	0					00000060	6590ef0a
		0a0abf8d	755a2be3	1f5449df	ff5b4307	33472d49	10444625
K-283	0				00d5d05a	1b6c5ace	e76b8ee3
		f925a572	19bcb952	12945154	588d0415	a5b4bb50	57f69216
K-409	0						0024ef90
		54eb3a6c	f4bdc6ed	021f6e5c	b8da0c79	5f913c52	ebaa9239
		8d1b7d3d	0adb8a34	add81800	acf7e302	a7d25095	1701d7a4
K-571	0	01cc6c27	e62f3e0d	df5ea7eb	1ab1cc4d	0da631c0	d70a969a
		a14b0350	85b31511	f5a97455	20cba528	e2d1e647	f4f708d3
		9fba0c3b	e4e35543	821344d1	662727bd	2d59dbc0	5e6853b1

to recover the integer equivalent. Solinas [4] showed that given the recurrence relation

$$U_{i+1} = \mu U_i - 2U_{i-1} \text{ where } U_0 = 0, U_1 = 1 \ ,$$

it is true that

$$\tau^i = \tau U_i - 2U_{i-1} \text{ for all } i > 0 \ .$$

Solinas noted that this equation can be used for computing the order of the curve. In addition to the aforementioned application, it is clear that this equation can also be directly applied to compute the equivalent element $d_0 + d_1\tau \in \mathbb{Z}[\tau]$ of a τ-adic expansion. Once $d_0 + d_1\tau$ is computed, the equivalent integer n modulo r is easily obtained using $n = d_0 + d_1 s \pmod r$.

We present an efficient algorithm to compute d_0 and d_1 as Algorithm 3. Note that the values d_0 and d_1 are built up in a right-to-left fashion. Clearly it makes sense to generate the fixed U sequence right-to-left. So if the coefficients are generated left-to-right and are not stored then U should be precomputed and stored and given as input to Algorithm 3, which is then modified to run

Algorithm 3. Integer equivalents of τ-adic expansions.

```
Input: $\ell$-bit $\tau$-adic expansion $\epsilon$, curve constants $r, s, \mu$
Output: Integer equivalent $n$
$d_0 \leftarrow \epsilon_0$, $d_1 \leftarrow \epsilon_1$                    /* accumulators */
$j \leftarrow 1$, $k \leftarrow 0$                          /* $j = U_{i-1}$, $k = U_{i-2}$ */
for $i \leftarrow 2$ to $\ell - 1$ do
    $u \leftarrow \mu j - 2k$                                   /* $u = U_i$ */
    $d_0 \leftarrow d_0 - 2j\epsilon_i$
    $d_1 \leftarrow d_1 + u\epsilon_i$
    $k \leftarrow j$, $j \leftarrow u$                       /* setup for next round */
end
$n \leftarrow d_0 + d_1 s \bmod r$                      /* integer equivalent */
return $n$
```

left-to-right. The storage of ϵ is small; indeed it can be stored in ℓ-bits if zeros inserted to preserve non-adjacency are omitted. Either way, the choice of which direction to implement Algorithm 3 is dependent on many factors, including storage capacity and direction of the scalar multiplication algorithm. In any case, the main advantage we are concerned with in this paper is the ability to compute the integer equivalent in parallel to the scalar multiple, and hence in our implementations ϵ is assumed to be stored. Thus, we omit any analysis of generating coefficients of ϵ one at a time, and concentrate on the scenario of having two separate devices, each with access to the coefficients of ϵ: one for computing the integer equivalent, and one for computing the scalar multiple.

As s is a per-curve constant, it is clear that the integer equivalent n can be computed using one field multiplication and one field addition. This excludes the cost of building up $d_0 + d_1\tau$ from the τ-adic expansion, done as shown in Algorithm 3 using the U sequence with only additions.

For the sake of simplicity, only width-2 τ-adic NAFs are considered here (all $\epsilon_i \in \{0, 1, -1\}$). Since generators are fixed and precomputation can be done offline, it is natural to consider arbitrary window width as well; we defer this to future work.

Following this reasoning, our method can be summarized as follows.

1. Generate a random τ-adic NAF of length $m - 1$. (After this, the scalar multiple can be computed in parallel to the remaining steps.)
2. Build up $d_0 + d_1\tau$ from the expansion using the U sequence as shown in Algorithm 3.
3. Calculate the integer equivalent $n = d_0 + d_1 s \pmod{r}$.

4 Hardware Implementation

If scalar multiplication is computed with a random integer, the integer is typically first reduced modulo δ. The τ-adic NAF then usually needs to be computed *before* scalar multiplication because algorithms for producing the τ-adic NAF presented in [4] produce them from right-to-left[1], whereas scalar multiplication is typically more efficient when computed from left-to-right. In either case, the end-to-end computation time is $T_\tau + T_{sm}$ where T_τ is the conversion time (including reducing modulo δ and generating the τ-adic NAF) and T_{sm} is the scalar multiplication time. However, when scalar multiplication is computed on hardware with a random τ-adic NAF of length ℓ, the calculation of an integer equivalent can be performed *simultaneously* with scalar multiplication (assuming separate dedicated hardware) thus resulting in an end-to-end time of only $\max(T_\tau, T_{sm}) = T_{sm}$ with the reasonable assumption that $T_{sm} > T_\tau$. In this case, T_τ denotes the time needed to generate a random τ-adic NAF and calculate the integer equivalent. We assume storage for the coefficients ϵ_i exists. This parallelization implies that our method is especially well-suited for hardware implementations.

[1] There are alternatives which produce an expansion with similar weight from right-to-left, but this does not change the arguments that follow.

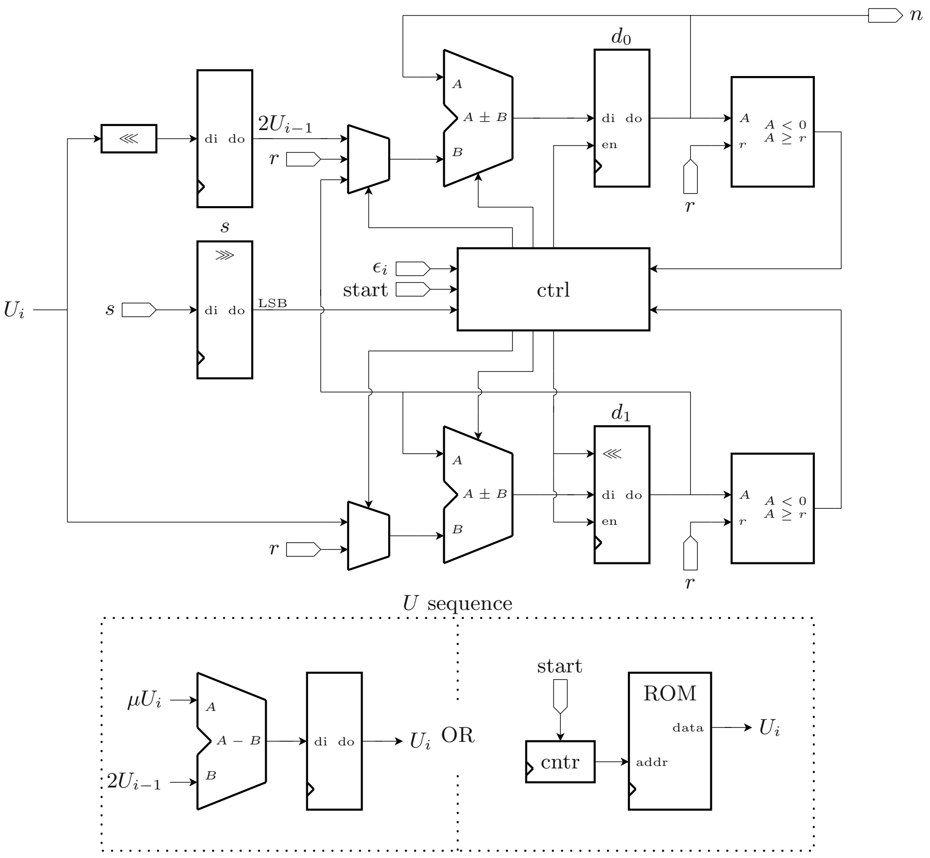

Fig. 1. Block diagram of the design. All registers are reset to zero at the beginning of a conversion, except the $2U_{i-1}$ register which is initialized to -1. Shifts to the left and to the right are denoted as $\lll$ and $\ggg$, respectively.

An FPGA design was implemented in order to investigate the practical feasibility of our method on hardware. The implementation consists of two adders, two comparators, a U sequence block and certain control logic. The structure of the implementation is shown in Fig. 1. An integer equivalent is computed so that, first, d_0 and d_1 are built up using the U sequence and, second, $n = d_0 + d_1 s$ is calculated as shown in Sec. 3. The design operates in two modes.

The first mode computes d_0 and d_1 in parallel using the adders. A τ-adic expansion is input into the design in serial starting from ϵ_0 and the U sequence is either read from ROM or computed on-the-fly. As shown in Sec. 3, U_i can be directly applied in computing d_1 by simply adding or subtracting U_i to d_1 according to ϵ_i. In d_0 calculation, $2U_{i-1}$ is received by shifting U_i to the left and delaying it by one clock cycle. Because the least significant bit (LSB) ϵ_0 is handled similarly, an additional value $U_{-1} = -0.5$ is introduced into the U sequence in order to get $d_0 = \epsilon_0$.

Table 2. ROM sizes for the NIST curves

Curve	m	Depth	Width	Bits
K-163	163	163	83	13,529
K-233	233	233	118	27,494
K-283	283	283	143	40,469
K-409	409	409	204	83,436
K-571	571	571	287	163,877

If the U sequence is precomputed and stored in ROM, the required size of the ROM depends on m. The depth of the ROM is m and the width is determined by the longest U_i in the sequence. ROM sizes for the NIST curves [9] are listed in Table 2. It makes sense not to reduce the sequence U modulo r, as $r > 1 + 2\sum_{i=0}^{m-3} |U_i|$ and hence U modulo r requires significantly more storage space than U alone.

If the amount of memory is an issue, the U sequence can be computed on-the-fly by using an adder as shown in Algorithm 3. This implies that extra storage for the coefficients ϵ_i is also needed for performing the scalar multiplication simultaneously. The width of the adder is also determined by the longest U_i. As shown in Table 2, the size of the ROM grows rapidly with m and in practice ROMs can be used only when m is small. However, the on-the-fly computation is also a viable approach for large m.

The second mode computes $n = d_0 + d_1 s$ in two phases which are repeated for all bits of s. In the first phase, the first adder accumulates d_0 with d_1 according to the LSB of s and, at the same time, d_1 is shifted to the left resulting in $2d_1$ and s is shifted to the right. In the second phase, r is either added to $d_{0,1}$ if $d_{0,1} < 0$ or subtracted from $d_{0,1}$ if $d_{0,1} \geq r$. This ensures that both d_0 and d_1 are in the interval $[0, r[$. When all bits of s have been processed, the register d_0 holds n. In order to guarantee that the procedure results in $n \in [0, r[$, r must fulfill $r > 1 + 2\sum_{i=0}^{m-3} |U_i|$ which is the maximum value of d_0 after the first mode. This ensures that $d_{0,1} \in]-r, r[$ in the end of the first mode for all τ-adic expansions with $\ell \leq m - 1$ and $\epsilon_i \in \{0, 1, -1\}$. It is easy to check that all r listed in [9] fulfill this condition.

The first mode requires $\ell + 1$ clock cycles. Because both phases in the second mode require one clock cycle and the length of s is m bits, the latency of the second phase is $2m$ clock cycles. Thus, the latency of a conversion is exactly $\ell + 2m + 1$ clock cycles where ℓ is the length of the τ-adic expansion. As $\ell = m - 1$, the conversion requires $3m$ clock cycles. The design is inherently resistant against side-channel attacks based on timing because its latency is constant.

If the U sequence is stored in ROM, it would be possible to reduce the latency by on average $\frac{2}{3}m$ clock cycles by skipping all zeros in random τ-adic NAFs in the first mode. This could also be helpful in thwarting side-channel attacks based on power or electromagnetic measurements. Unfortunately, latency would not be constant anymore making the design potentially insecure against timing attacks. Reductions based on zero skippings are not possible if the U sequence is computed on-the-fly because that computation always requires m clock cycles.

4.1 Results

The design presented in Sec. 4 was written in VHDL and simulated in ModelSim SE 6.1b. To the best of our knowledge, the only published hardware implementation of the integer to τ-adic NAF conversion was presented in [11] where the converter was implemented on an Altera Stratix II EP2S60F1020C4 FPGA. In order to ensure fair comparison, we synthesized our design for the same device using Altera Quartus II 6.0 SP1 design software. Two curve-specific variations of the design were implemented for the NIST curves K-163 and K-233.

The K-163 design with ROM requires 929 ALUTs (Adaptive Look-Up Tables) and 599 registers in 507 ALMs (Adaptive Logic Modules) and 13,529 bits of ROM which were implemented by using 6 M4K memory blocks. The maximum clock frequency is 56.14 MHz which yields the conversion time of 8.7 μs. The K-233 design with ROM has the following characteristics: 1,311 ALUTs and 838 registers in 713 ALMs, 27,612 memory bits in 7 M4Ks and 42.67 MHz resulting in 16.4 μs. Implementations where the U sequence is computed on-the-fly require 1,057 and 1,637 ALUTs and 654 and 934 registers in 592 and 894 ALMs for K-163 or K-233, respectively. They operate at the maximum clock frequencies of 55.17 and 43.94 MHz resulting in the computation times of 8.9 μs and 15.9 μs. The differences in computation times compared to the ROM-based implementations are caused by small variations in place&route results which yield slightly different maximum clock frequencies. The latencies in clock cycles are the same for both ROM-based and memory-free implementations, i.e. 489 for K-163 and 699 for K-233. As the required resources are small and the conversion times are much shorter than any reported scalar multiplication time, our method is clearly suitable for hardware implementations.

4.2 Comparisons

The implementation presented in [11] computes τ-adic NAF mod $(\tau^m - 1)$ so that it first converts the integer to τ-adic NAF with an algorithm from [3], then reduces it modulo $(\tau^m - 1)$ and finally reconstructs the NAF which was lost in the reduction. This was claimed to be more efficient in terms of required resources than reductions modulo δ presented in [3] because their implementation is problematic on hardware as they require computations of several multiplications, and hence either a lot of resources or computation time.

Table 3 summarizes the implementations presented here and in [11]. Comparing the implementations is straightforward because FPGAs are the same. It should be noted that the converter in [11] has a wider scope of possible applications since our approach is only for taking random multiples of a point (this is not useful in signature verifications, for example). Obviously, the computation of an integer equivalent can be performed with fewer resources. The reductions are 35 % in ALUTs and 39 % in registers for K-163 and 27 % and 30 % for K-233 when the U sequence is stored in the ROM. However, it should be noted that the implementations presented in [11] do not require such additional memory. When the U sequence is computed with logic and no ROM is needed, the reductions in

Table 3. Comparison of the published designs on Stratix II FPGA

Design	m	ALUTs	Regs.	M4Ks	Clock (MHz)	T_τ (μs)	Total time
[11][a]	163	1,433	988	0	80.44	6.1	$T_{sm} + T_\tau$
	233	1,800	1,198	0	58.89	11.9	$T_{sm} + T_\tau$
This work[b]	163	929 (-35 %)	599 (-39 %)	6	56.14	8.7	T_{sm}
	163	1,057 (-26 %)	654 (-34 %)	0	55.17	8.9	T_{sm}
	233	1,311 (-27 %)	838 (-30 %)	7	42.67	16.4	T_{sm}
	233	1,637 (-9 %)	934 (-22 %)	0	43.94	15.9	T_{sm}

[a] Integer to τ-adic NAF conversion.
[b] τ-adic expansion to integer conversion.

ALUTs and registers are 26 % and 34 % for K-163 and 9 % and 22 % for K-233 and so it is obvious that our converter is more compact.

The average latencies of both converters are approximately the same. The difference is that the latency of our converter is always exactly 489 or 699 clock cycles whereas the converter in [11] has an average latency of 491 or 701 clock cycles for K-163 and K-233, respectively. The maximum clock frequencies of our converters are lower and, thus, the implementations of [11] can compute conversions faster. However, an integer equivalent can be computed in parallel with scalar multiplication and, thus, it can be claimed that the effective conversion time is 0 μs.

To support the argument that the effective elimination of the conversion time T_τ is significant, there are several implementations existing in the literature computing scalar multiplications on K-163 in less than 100 μs. For example, [12] reports a time of 44.8 μs, and [13] a time of 75 μs. Hence conversions requiring several μs are obviously significant when considering the overall time.

To summarize, computing the integer equivalent of a random τ-adic expansion offers the following two major advantages from the hardware implementation point-of-view compared to computing the τ-adic NAF of a random integer:

- Conversions can be computed in parallel with scalar multiplications.
- Computing the integer equivalent can be implemented with fewer resources.

As a downside, the calculation of an integer equivalent has a longer latency; however, this is insignificant since the conversion is not on the critical path.

5 Conclusion

As shown, our new method for computing integer equivalents of random τ-adic expansions is very computationally efficient. This has been demonstrated with an implementation in hardware, where the parallelization of computing the integer equivalent and the scalar multiple yields significant efficiency gains. It seems unlikely that such gains are possible with this approach in software.

Future Work

Side-channel attacks based on timing, power, or electromagnetic measurements are a serious threat to many implementations; not only on smart cards, but on FPGAs [14] as well. Our converter provides inherent resistance against timing attacks because its latency is constant. Side-channel countermeasures against other attacks are beyond the scope of this paper. However, before the suggested implementation can be introduced in any practical application where these attacks are viable, it must be protected against such attacks. This will be an important research topic in the future.

As mentioned, only width-2 τ-adic NAFs have been considered here (all $\epsilon_i \in \{0, 1, -1\}$). Arbitrary window width would clearly be more efficient for the scalar multiplication. We are currently researching efficient methods for scanning ϵ multiple bits at once, as well as simple alternatives to using the U sequence.

Acknowledgements

We thank Tanja Lange for useful discussions and comments during the preparation of this paper. We also thank the anonymous reviewers for comments on the manuscript.

References

1. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
2. Meier, W., Staffelbach, O.: Efficient multiplication on certain nonsupersingular elliptic curves. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 333–344. Springer, Heidelberg (1993)
3. Solinas, J.A.: An improved algorithm for arithmetic on a family of elliptic curves. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 357–371. Springer, Heidelberg (1997)
4. Solinas, J.A.: Efficient arithmetic on Koblitz curves. Des. Codes Cryptogr. 19(2-3), 195–249 (2000)
5. Lange, T., Shparlinski, I.: Collisions in fast generation of ideal classes and points on hyperelliptic and elliptic curves. Appl. Algebra Engrg. Comm. Comput. 15(5), 329–337 (2005)
6. Lange, T., Shparlinski, I.E.: Certain exponential sums and random walks on elliptic curves. Canad. J. Math. 57(2), 338–350 (2005)
7. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Information Theory IT-22(6), 644–654 (1976)
8. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
9. National Institute of Standards and Technology (NIST): Digital signature standard (DSS). Federal Information Processing Standard, FIPS PUB 186-2 (2000)
10. Lange, T.: Koblitz curve cryptosystems. Finite Fields Appl. 11(2), 200–229 (2005)

11. Järvinen, K., Forsten, J., Skyttä, J.: Efficient circuitry for computing τ-adic non-adjacent form. In: ICECS 2006. Proc. of the IEEE Int'l. Conf. on Electronics, Circuits and Systems, Nice, France, pp. 232–235 (2006)
12. Dimitrov, V.S., Järvinen, K.U., Jacobson, J.M.J., Chan, W.F., Huang, Z.: FPGA implementation of point multiplication on Koblitz curves using Kleinian integers. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 445–459. Springer, Heidelberg (2006)
13. Lutz, J., Hasan, M.A.: High performance FPGA based elliptic curve cryptographic co-processor. In: Goubin, L., Matsui, M. (eds.) ITCC 2004. International Conference on Information Technology: Coding and Computing, vol. 02, pp. 486–492. IEEE Computer Society Press, Los Alamitos (2004)
14. Standaert, F.X., Peeters, E., Rouvroy, G., Quisquater, J.J.: An overview of power analysis attacks against field programmable gate arrays. Proc. IEEE 94(2), 383–394 (2006)