J Cryptogr Eng (2011) 1:187–199
DOI 10.1007/s13389-011-0017-8

REGULAR PAPER

Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction

Jonathan Taverne · Armando Faz-Hernández · Diego F. Aranha · Francisco Rodríguez-Henríquez · Darrel Hankerson · Julio López

Received: 2 September 2011 / Accepted: 8 September 2011 / Published online: 25 September 2011

Abstract The availability of a new carry-less multiplication instruction in the latest Intel desktop processors significantly accelerates multiplication in binary fields and hence presents the opportunity for reevaluating algorithms for binary field arithmetic and scalar multiplication over elliptic curves. We describe how to best employ this instruction in field multiplication and the effect on performance of doubling and halving operations. Alternate strategies for implementing inversion and half-trace are examined to restore most of their competitiveness relative to the new multiplier. These improvements in field arithmetic are complemented by a study on serial and parallel approaches for Koblitz and random curves, where parallelization strategies are implemented and compared. The contributions are illustrated with experimental results improving the state-of-the-art performance of halving and doubling-based scalar multiplication on NIST curves at the 112- and 192-bit security levels and a new speed record for side-channel-resistant scalar multiplication in a random curve at the 128-bit security level. The algorithms presented in this work were implemented on Westmere and Sandy Bridge processors, the latest generation Intel microarchitectures.

Keywords Elliptic curve cryptography · Finite field arithmetic · Parallel algorithm · Efficient software implementation

J. Taverne: This work was performed while the author was visiting CINVESTAV-IPN. D. F. Aranha: A portion of this work was performed while the author was visiting University of Waterloo.

J. Taverne
Université de Lyon, Université Lyon 1, ISFA, Lyon, France
e-mail: jonathan.taverne@etu.univ-lyon1.fr

A. Faz-Hernández · F. Rodríguez-Henríquez (✉)
Computer Science Department, CINVESTAV-IPN,
Mexico City, Mexico
e-mail: francisco@cs.cinvestav.mx

A. Faz-Hernández
e-mail: armfaz@computacion.cs.cinvestav.mx

D. F. Aranha · J. López
Institute of Computing, University of Campinas,
Campinas, Brazil
e-mail: dfaranha@ic.unicamp.br

J. López
e-mail: jlopez@ic.unicamp.br

D. Hankerson
Auburn University, Auburn, USA
e-mail: hankedr@auburn.edu

1 Introduction

Improvements in the fabrication process of microprocessors allow the resulting higher transistor density to be converted into architectural features such as inclusion of new instructions or faster execution of the current instruction set. Limits on the conventional ways of increasing a processor's performance such as incrementing the clock rate, scaling the memory hierarchy [44] or improving support for instruction-level parallelism [43] have pushed manufacturers to embrace parallel processing as the mainstream computing paradigm and consequently amplify support for resources such as multiprocessing and vectorization. Examples of the latter are the recent inclusions of the SSE4 [23], AES [19] and AVX [14] instruction sets in the latest Intel microarchitectures.

Since the dawn of elliptic curve cryptography in 1985, several field arithmetic assumptions have been made by researchers and designers regarding its efficient implementation in software platforms. Some analysis (supported by experiments) assumed that inversion to multiplication ratios

(I/M) were sufficiently small (e.g., $I/M \approx 3$) that point operations would be done in affine coordinates, favoring certain techniques. However, the small ratios were a mix of old hardware designs, slower multiplication algorithms compared with [36] and composite extension degree. It seems clear that sufficient progress was made in multiplication so there is incentive to use projective coordinates. Our interest in the face of a much faster multiplication is at the other end—is I/M large enough to affect methods that commonly assumed this ratio is modest?

On the other hand, authors in [16] considered that the cost of a point halving computation was roughly equivalent to 2 field multiplications. The expensive computations in halving are a field multiplication, solving a quadratic $z^2+z=c$, and finding a square root over $\mathbb{F}_{2^m}$. However, quadratic solvers presented in [21] are multiplication-free and hence, provided that a fast binary field multiplier is available, there would be concern that the ratio of point halving to multiplication may be much larger than 2. Having a particularly fast multiplier would also push for computing square roots in $\mathbb{F}_{2^m}$ as efficiently as possible. Similarly, the common software design assumption that field squaring is essentially free (relative to multiplication) may no longer be valid.

A prevalent assumption is that large-characteristic fields are faster than binary field counterparts for software implementations of elliptic curve cryptography.[1] In spite of simpler arithmetic, binary field realizations could not be faster than large-characteristic analogs mostly due to the absence of a native *carry-less multiplier* in contemporary high-performance processors. However, using a bit-slicing technique, Bernstein [6] was able to compute a batch of 251-bit scalar multiplications on a binary Edwards curve, employing 314,323 clock cycles per scalar multiplication, which, before the results presented in this work and to the best of our knowledge, was the fastest reported time for a software implementation of binary elliptic point multiplication.

In this work, the impact of the recently introduced carry-less multiplication instruction [20] in the performance of binary field arithmetic and scalar multiplication over elliptic curves is evaluated. We also consider parallel strategies in order to speed scalar multiplication when working on multi-core architectures. In contrast to parallelization applied to a batch of operations, the approach considered here applies to a single point multiplication. These approaches target different environments: batching makes sense when throughput is the measurement of interest, while the lower level parallelization is of interest when latency matters and the device is perhaps weak but has multiple processing units. Furthermore, throughout this paper it will be assumed that all the computations are done in the unknown point scenario, i.e., where the elliptic curve point to be processed is not known in advance, thus precluding off-line precomputation. We will assume that there is sufficient memory space for storing a few multiples of the point to be processed and look-up tables for accelerating the computation of the underlying field arithmetic.

As the experimental results will show, our implementation of multiplication via this native support was significantly faster than previous timings reported in the literature. This motivated a study on alternative implementations of binary field arithmetic in hope of restoring the performance ratios among different operations in which the literature is traditionally based [21]. A direct consequence of this study is that performance analysis based on these conventional ratios [5] will remain valid in the new platform. Our main contributions are

- A strategy to efficiently employ the native carry-less multiplier in binary field multiplication.
- Branchless and/or vectorized approaches for implementing half-trace computation, integer recoding and inversion. These approaches allow the halving operation to become again competitive with doubling in the face of a significantly faster multiplier and help to reduce the impact of integer recoding and inversion in the overall speed of scalar multiplication, even when projective coordinates are used.
- Parallelization strategies for dual-core execution of scalar multiplication algorithms in random and Koblitz binary elliptic curves.

We obtain a new state-of-the-art implementation of arithmetic in binary elliptic curves, including improved performance for NIST-standardized Koblitz curves and random curves suitable for halving and a new speed record for side-channel resistant point multiplication in a random curve at the 128-bit security level.

This paper reports a software library that performs scalar multiplication over the NIST Koblitz and random binary curves K-233, B-233, K-409 and B-409 in 89, 182, 321 and 705 thousand cycles, respectively, on a single core of a 3.326 GHz Intel Westmere Core i5 660 processor. Our library was also adjusted to compute scalar multiplication in a 251-bit binary Edwards curve in 282 thousand clock cycles. Moreover, we also introduce parallel strategies that accelerate scalar multiplication by using both cores present in the processor. In this setting, scalar multiplication on K-233, B-233, K-409 and B-409 curves is computed in 58, 116, 191 and 444 thousand cycles, respectively. These dual-core timings imply, for example, that scalar multiplication on K-233 can be computed in less than 17.5 µs.

[1] In hardware realizations, the opposite thesis is widely accepted: elliptic curve scalar point multiplication can be computed (much) faster using binary extension fields.

Experimental results of the implementation of our library on a 3.4 GHz Intel Sandy Bridge Core i7 2600K processor are also provided. Taking advantage of the 256-bit registers with new addressing mode capabilities of this processor, timings for NIST Koblitz and random binary curves K-233, B-233, K-409 and B-409 are reported at a cost of 68, 157, 256 and 557 thousand cycles, respectively. Scalar multiplication in a 251-bit binary Edwards curve can be computed in just 225 thousand clock cycles. Finally, using a parallel approach scalar multiplication on K-233, B-233, K-409 and B-409 is performed at a cost of 47, 100, 149 and 349 thousand cycles, respectively, using two cores of the Sandy Bridge processor.[2]

The remainder of the paper progresses as follows: Section 2 elaborates on exploiting carry-less multiplication for high-performance field multiplication along with implementation strategies for half-trace and inversion. Sections 3 and 4 discuss serial and parallel approaches for scalar multiplication. Section 5 presents extensive experimental results and comparison with related work. Section 6 concludes the paper with perspectives on the interplay between the proposed implementation strategies and future enhancements in the architecture under consideration.

2 Binary field arithmetic

A binary extension field $\mathbb{F}_{2^m}$ can be constructed by means of a degree-m polynomial f irreducible over $\mathbb{F}_2$ as $\mathbb{F}_{2^m} \cong \mathbb{F}_2[z]/(f(z))$. In the case of software implementations in modern desktop platforms, field elements $a \in \mathbb{F}_{2^m}$ can be represented as polynomials of degree at most $m-1$ with binary coefficients a_i packed in $n_{64} = \lceil \frac{m}{64} \rceil$ 64-bit processor words. In this context, the recently introduced carry-less multiplication instruction can play a significant role in order to efficiently implement a multiplier in $\mathbb{F}_{2^m}$. Along with field multiplication, other relevant field arithmetic operations such as squaring, square root and half-trace will be discussed in the rest of this section.

2.1 Multiplication

Field multiplication is the performance-critical operation for implementing several cryptographic primitives relying on binary fields, including arithmetic over elliptic curves and the Galois Counter Mode of operation (GCM). For accelerating the latter when used in combination with the AES block cipher [19], Intel introduced the carry-less multiplier in the Westmere microarchitecture as an instruction operating on 64-bit words stored in 128-bit vector registers with opcode *pclmulqdq* [20]. The instruction latency currently peaks at 15 cycles while reciprocal throughput ranks at 10 cycles. In other words, when operands are not in a dependency chain, effective latency is 10 cycles [15].

The instruction certainly looks expensive when compared with the 3-cycle 64-bit integer multiplier present in the same platform, which raises speculation whether Intel aimed for an area/performance trade-off or simply balanced the latency to the point where the carry-less multiplier did not interfere with the throughput of the hardware AES implementation. Either way, the instruction features suggest the following empirical guidelines for organizing the field multiplication code: (i) as memory access by vector instructions continues to be expensive [6], the maximum amount of work should be done in registers, for example, through a Comba organization [12]; (ii) as the number of registers employed in multiplication should be minimized for avoiding false dependencies and maximize throughput, the multiplier should have 128-bit granularity; (iii) as the instruction latency allows, each 128-bit multiplication should be implemented with three carry-less multiplications in a Karatsuba fashion [28].

In fact, the overhead of Karatsuba multiplication is minimal in binary fields and the Karatsuba formula with the smaller number of multiplications for multiplying $\lceil \frac{n_{64}}{2} \rceil$ 128-bit digits proved to be optimal in all the considered field sizes. This observation comes in direct contrast to previous vectorized implementations of the *comb* method for binary field multiplication due to López and Dahab [36, Algorithm 5], where the memory-bound precomputation step severely limits the number of Karatsuba steps which can be employed, fixing the cutoff point to large fields [2] such as $\mathbb{F}_{2^{1223}}$. To summarize, multiplication was implemented as a 128-bit granular Karatsuba multiplier with each 128-digit multiplication solved by another Karatsuba instance requiring three carry-less multiplications, cheap additions and efficient shifts by multiples of 8 bits. A single 128-digit level of Karatsuba was used for fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{251}}$ where $\lceil \frac{n_{64}}{2} \rceil = 2$, while two instances were used for field $\mathbb{F}_{2^{409}}$ where $\lceil \frac{n_{64}}{2} \rceil = 4$. Particular approaches which led to lower performance in our experiments were organizations based on optimal Toom-Cook [10] due to the higher overhead brought by minor operations and on a lower 64-bit granularity combined with alternative multiple-term Karatsuba formulas [38] due to register exhaustion to store all the intermediate values, causing a reduction in overall throughput.

2.2 Squaring, square-root and multi-squaring

Squaring and square-root are considered cheap operations in a binary field, especially when $\mathbb{F}_{2^m}$ is defined by a square-root friendly polynomial [1,3], because they require only linear manipulation of individual coefficients [21]. These operations are traditionally implemented with the help of

[2] Preliminary results reported here have been published in [42]; this paper is an enhanced and extended version.

large precomputed tables, but vectorized implementations are possible with simultaneous table lookups through byte shuffling instructions [2]. This approach is enough to keep square and square-root efficient relative to multiplication even with a dramatic acceleration of field multiplication. For illustration, Aranha et al. [2] reports multiplication-to-squaring ratios as high as 34 without a native multiplier, far from the conventional ratios of 5 [5] or 7 [21] and with a large room for future improvement.

Multi-squaring, or exponentiation to 2^k, can be efficiently implemented with a time-memory trade-off proposed as m-squaring in [1,11] and here referred as *multi-squaring*. For a fixed k, a table T of $16\lceil\frac{m}{4}\rceil$ field elements can be precomputed such that $T[j, i_0 + 2i_1 + 4i_2 + 8i_3] = (i_0 z^{4j} + i_1 z^{4j+1} + i_2 z^{4j+2} + i_3 z^{4j+3})^{2^k}$ and $a^{2^k} = \sum_{j=0}^{\lceil\frac{m}{4}\rceil} T[j, \lfloor a/2^{4j} \rfloor \bmod 2^4]$. The threshold where multi-squaring became faster than simple consecutive squaring observed in our implementation was around $k \geq 6$ for $\mathbb{F}_{2^{233}}$ and $k \geq 10$ for $\mathbb{F}_{2^{409}}$.

2.3 Inversion

Inversion modulo $f(z)$ can be implemented via the polynomial version of the Extended Euclidean Algorithm (EEA), but the frequent branching and recurrent shifts by arbitrary amounts present a performance obstacle for vectorized implementations, which makes it difficult to write consistently fast EEA codes across different platforms. A branchless approach can be implemented through Itoh-Tsuji inversion [24] by computing $a^{-1} = a^{(2^{m-1}-1)2}$, as proposed in [18]. In contrast to the EEA method, the Itoh-Tsujii approach has the additional merit of being similarly fast (relative to multiplication) across common processors.

The overall cost of the method is $m - 1$ squarings and a number of multiplications dictated by the length of an addition chain for $m - 1$. The cost of squarings can be reduced by computing each required 2^i-power as a multi-squaring [11]. The choice of an addition chain allows the implementer to control the amount of required multiplications and the precomputed storage for multi-squaring, since the number of 2^i-powers involved can be balanced.

A previous work obtained inversion-to-multiplication ratios between 22 and 41 by implementing EEA in 64-bit mode [2], while the conventional ratios are between 5 and 10 [5,21]. While we cannot reach the small ratios with Itoh-Tsujii for the parameters considered here, we can hope to do better than applying the method from [2] which will give significantly larger ratios with the carry-less multiplier. Hence, the cost of squarings and multi-squarings should be minimized to the lowest possible allowed by storage capacity.

To summarize, we use addition chains of 10, 10 and 11 steps for computing field inversion over the fields $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{251}}$ and $\mathbb{F}_{2^{409}}$, respectively.[3] We extensively used the multi-squaring approach described in the preceding section. For example, in the case of $\mathbb{F}_{2^{233}}$, the addition chain $1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 28 \to 29 \to 58 \to 116 \to 232$ was selected and used 3 pre-computed tables for computing the iterated squarings $a^{2^{29}}$, $a^{2^{58}}$ and $a^{2^{116}}$. The rest of the field squaring operations were computed by executing consecutive squarings. Let us recall that each table stores a total of $16\lceil\frac{m}{4}\rceil$ field elements.

2.4 Half-trace

Half-trace plays a central role in point halving and its performance is essential if halving is to be competitive against doubling. For an odd integer m, the half-trace function $H : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ is defined by $H(c) = \sum_{i=0}^{(m-1)/2} c^{2^{2i}}$ and satisfies the equation $\lambda^2 + \lambda = c + \mathrm{Tr}(c)$ required for point halving. One efficient desktop-targeted implementation of the half-trace is described in [3] and presented as Algorithm 1, making extensive use of precomputations. This implementation is based on two main steps: the elimination of even power coefficients and the accumulation of half-trace precomputed values.

Step 5 in Algorithm 1, as shown in [21], consists in reducing the number of non-zero coefficients of c by removing the coefficients of even powers i by means of the identity $H(z^i) = H(z^{i/2}) + z^{i/2} + \mathrm{Tr}(z^i)$. That will lead to memory and time savings during the last step of the half-trace computation, the accumulation part (step 6). This is done by extraction of the odd and even bits and can benefit from vectorization in the same way as square-root in [2]. However, in the case of half-trace there is a bottleneck caused by data dependencies. For efficiency, the bank of 128-bit registers is used as much as possible, but at one point in the algorithm execution the number of available bits to process decreases. For 64-bit and 32-bit digits, the use of 128-bit registers is still beneficial, but for a smaller size, the conventional approach (not vectorized) becomes again competitive.

Unlike the direction taken in [21], the approach in [3] does not attempt to minimize memory requirements but rather it greedily strives to speed the accumulation part (step 6). Precomputation is extended so as to reduce the number of accesses to the lookup table. The following values of the half-trace are stored: $H(l_0 c^{8i+1} + l_1 c^{8i+3} + l_2 c^{8i+5} + l_3 c^{8i+7})$ for all $i \geq 0$ such that $8i < m - 3$ and $l_j \in \mathbb{F}_2$. The memory size in bytes taken by the precomputations follows the formula $16 \times n_{64} \times 8 \times \lceil\frac{m}{8}\rceil$.

[3] In the case of inversion over $\mathbb{F}_{2^{409}}$, the minimal length addition chain to reach $m - 1 = 408$ has 10 steps. However, an 11-step chain was preferred in order to save one look-up table.

Algorithm 1 Solve $x^2 + x = c$

```
Input: $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$ where $m$ is odd and $\mathrm{Tr}(c) = 0$
Output: a solution $s$ of $x^2 + x = c$
1: compute $H(l_0 c^{8i+1} + l_1 c^{8i+3} + l_2 c^{8i+5} + l_3 c^{8i+7})$ for $i \in I = \{0, \ldots, \lfloor \frac{m-3}{8} \rfloor\}$ and $l_j \in \mathbb{F}_2$
2: $s \leftarrow 0$
3: for $i = (m-1)/2$ downto 1 do
4:   if $c_{2i} = 1$ then
5:     $c \leftarrow c + z^i, s \leftarrow s + z^i$
6: return $s \leftarrow s + \sum_{i \in I} c^{8i+1} H(z^{8i+1}) + c^{8i+3} H(z^{8i+3}) + c^{8i+5} H(z^{8i+5}) + c^{8i+7} H(z^{8i+7})$
```

While considering different organizations of the half-trace code, we made the following serendipitous observation: inserting as many xor operations as the data dependencies permitted from the accumulation stage (step 6) into step 5 gave a substantial speed-up of 20% to 25% compared with code written in the order as described in Algorithm 1. Plausible explanations are compiler optimization and processor pipelining characteristics. The result is a half-trace-to-multiplication ratio near 1, and this ratio can be reduced if memory can be consumed more aggressively.

3 Random binary elliptic curves

Given a finite field $\mathbb{F}_q$ for $q = 2^m$, a non-supersingular elliptic curve $E(\mathbb{F}_q)$ is defined to be the set of points $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ that satisfy the affine equation

$$y^2 + xy = x^3 + ax^2 + b, \qquad (1)$$

where a and $0 \neq b \in \mathbb{F}_q$, together with the point at infinity denoted by $\mathcal{O}$. It is known that $E(\mathbb{F}_q)$ forms an additive Abelian group with respect to the elliptic point addition operation.

Let k be a positive integer and P a point on an elliptic curve. Then *elliptic curve scalar multiplication* is the operation that computes the multiple $Q = kP$, defined as the point resulting of adding P to itself $k - 1$ times. One of the most basic methods for computing a scalar multiplication is based on a double-and-add variant of Horner's rule. As the name suggests, the two most prominent building blocks of this method are the *point doubling* and *point addition* primitives. By using the non-adjacent form (NAF) representation of the scalar k, the addition-subtraction method computes a scalar multiplication in about m doubles and $m/3$ additions [21]. The method can be extended to a *width-ω NAF* $k = \sum_{i=0}^{t-1} k_i 2^i$ where $k_i \in \{0, \pm 1, \ldots, \pm 2^m - 1\}$, $k_{t-1} \neq 0$, and at most one of any ω consecutive digits is nonzero. The length t is at most one larger than the bitsize of k, and the density is approximately $1/(\omega + 1)$; for $\omega = 2$, this is the same as NAF.

3.1 Sequential algorithms for random binary curves

The traditional left-to-right double-and-add method is illustrated in Algorithm 2, and the *width-ω NAF* $k = \sum_{i=0}^{t-1} k_i 2^i$ expression is computed from left to right, i.e., it starts processing k_{t-1} first, then k_{t-2} until it ends with the coefficient k_0. Step 1 computes $2^{\omega-2} - 1$ multiples of the point P. Based on the Montgomery trick, authors in [13] suggested a method to precompute the affine points in large-characteristic fields $\mathbb{F}_p$, employing only one inversion. Exporting that approach to $\mathbb{F}_{2^m}$, we obtained formulae that offer a saving of 4 multiplications and 15 squarings for $\omega = 4$ when compared with a naive method that would make use of the Montgomery trick in a trivial way (see Table 1 for a summary of the computational effort associated with this phase).

Algorithm 2 Double-and-add scalar multiplication

```
Input: $\omega, k, P \in E(\mathbb{F}_{2^m})$ of odd order $r$
Output: $kP$
1: Obtain the representation $\omega\mathrm{NAF}(k) = \sum_{i=0}^{t} k_i 2^i$
2: Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
3: $Q \leftarrow \mathcal{O}$
4: for $i = t$ downto 0 do
5:   $Q \leftarrow 2Q$
6:   if $k_i > 0$ then
7:     $Q \leftarrow Q + P_{k_i}$
8:   else if $k_i < 0$ then
9:     $Q \leftarrow Q - P_{-k_i}$
10: return $Q$
```

For a given ω, the evaluation stage of the algorithm has approximately $m/(\omega+1)$ point additions and hence increasing ω has diminishing returns. For the curves given by NIST [39] and with on-line precomputation, $\omega \leq 6$ is optimal in the sense that total point additions are minimized. In many cases, the recoding in $\omega\mathrm{NAF}(k)$ is performed on-line and can be considered as part of the precomputation step.

Table 1 Costs and parameter recommendations for $\omega \in \{3, 4, 5\}$

ω	Algorithm 4		Alg 5	Alg 6
	Precomp	Postcomp	Precomp	Postcomp
3	$14M$,$11S$,I	$43M$,$26S$	$2M$,$3S$,I	$26M$,$13S$
4	$38M$,$15S$,I	$116M$,$79S$	$9M$,$9S$,I	$79M$,$45S$
5	N/A	N/A	$23M$,$19S$,$2I$	$200M$,$117S$

(a) Pre- and post-computation costs.

ω	Algorithm 4		Algorithm 7	
	B-233	B-409	K-233	K-409
3	128	242	131	207
4	132	240	135	210
5	N/A	N/A	136	213

(b) Recommended value for n.

Table 2 Timings in clock cycles for field arithmetic operations on a Westmere processor

Base field operation	$\mathbb{F}_{2^{233}}$			$\mathbb{F}_{2^{251}}$			$\mathbb{F}_{2^{409}}$		
	GCC	ICC	op/M	GCC	ICC	op/M	GCC	ICC	op/M
Multiplication	128	128	1.00	161	159	1.00	345	348	1.00
López–Dahab Mult.	256	367	2.87	338	429	2.70	637	761	2.19
Square root	67	60	0.47	155	144	0.91	59	56	0.16
Squaring	30	35	0.27	56	59	0.37	44	49	0.14
Half trace	167	150	1.17	219	212	1.33	322	320	0.92
Multi-Squaring	191	184	1.44	195	209	1.31	460	475	1.36
Inversion	2,951	2,914	22.77	3,710	3,878	24.39	9,241	9,350	26.87
4-τNAF	9,074	11,249	87.88	–	–	–	23,783	26,633	76.53
3-NAF	5,088	5,059	39.52	–	–	–	13,329	14,373	41.30
4-NAF	4,280	4,198	32.80	–	–	–	11,406	12,128	34.85
Recoding (halving)	1,543	1,509	11.79	–	–	–	3,382	3,087	8.87
Recoding (parallel)	999	1,043	8.15	–	–	–	2,272	2,188	6.29

op/M denotes ratio to multiplication obtained from ICC

Table 3 Timings in clock cycles for curve arithmetic operations on a Westmere processor

Elliptic curve operations	B-233			B-409		
	GCC	ICC	op/M	GCC	ICC	op/M
Doubling (LD)	690	710	5.55	1,641	1,655	4.76
Addition (KIM Mixed)	1,194	1,171	9.15	2,987	3,000	8.62
Addition (LD Mixed)	1,243	1,233	9.63	3,072	3,079	8.85
Addition (LD General)	1,954	1,961	15.32	4,893	4,922	14.14
Halving	439	417	3.26	894	878	2.52

op/M denotes ratio to multiplication obtained from ICC

The most popular way to represent points in binary curves is López–Dahab projective coordinates that yield an effective cost for a mixed point addition and point doubling operation of about $8M + 5S \approx 9M$ and $4M + 5S \approx 5M$, respectively (see Tables 2 and 3). Kim and Kim [29] report alternate formulas for point doubling requiring four multiplications and five squarings, but two of the four multiplications are by the constant b, and these have the same cost as general multiplication with the native carry-less multiplier. For mixed addition, Kim and Kim require eight multiplications but save two field reductions when compared with López–Dahab, giving their method the edge. Hence, in this work López–Dahab was adopted for point doubling and Kim and Kim for point addition.

Right-to-left halve-and-add

Scalar multiplication based on point halving replaces point doubling by a potentially faster *halving* operation that produces Q from P with $P = 2Q$. The method was proposed independently by Knudsen [31] and Schroeppel [40] for curves $y^2 + xy = x^3 + ax^2 + b$ over $\mathbb{F}_{2^m}$. The method is simpler if the trace of a is 1, and this is the only case considered here. The expensive computations in halving are a field multiplication, solving a quadratic $z^2 + z = c$, and finding a square root. On the NIST random curves studied in this work, we found that the cost of halving is approximately $3M$, where M denotes the cost of a field multiplication. In [16] authors proposed Algorithm 3, a right-to-left halve-and-add method that allows efficient windowing computation and that is especially suitable in the case of large I/M ratios.

Let the base point P have odd order r, and let t be the number of bits to represent r. For $0 < n \leq t$, let $\sum_{i=0}^{t} k'_i 2^i$ be given by the width-ω NAF of $2^n k \bmod r$. Then $k \equiv k'/2^n \equiv \sum_{i=0}^{t} k'_i 2^{i-n} \pmod r$ and the scalar multiplication can be split as

$$kP = (k'_t 2^{t-n} + \cdots + k'_n)P + (k'_{n-1} 2^{-1} + \cdots + k'_0 2^{-n})P. \tag{2}$$

When $n = t$, this gives the usual representation for point multiplication via halving, illustrated in Algorithm 4 (that is,

Algorithm 3 Halve-and-add scalar multiplication

```
Input: $\omega, k, P \in E(\mathbb{F}_{2^m})$ of odd order $r$
Output: $kP$
Perform scalar recoding: $k' = 2^t k \bmod r$ where $t = \lceil \log_2 r \rceil$
Obtain the representation $\omega\mathrm{NAF}(k')/2^t = \sum_{i=0}^{t} k'_i 2^{i-t}$
Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
for $i = t$ downto 0 do
    if $k'_i > 0$ then
        $Q_{k'_i} \leftarrow Q_{k'_i} + P$
    else if $k'_i < 0$ then
        $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
    $P \leftarrow P/2$
return $Q \leftarrow \sum_{i \in I} i Q_i$
```

the computation is essentially the right column). The cost for postcomputation appears in Table 1.

3.2 Parallel scalar multiplication on random binary curves

For parallelization, choose $n < t$ in (2) and process the first portion by a double-and-add method and the second portion by a method based on halve-and-add. Algorithm 4 illustrates a parallel approach suitable for two processors. Recommended values for n to balance cost between processors appear in Table 1.

Algorithm 4 Double-and-add, halve-and-add scalar multiplication: parallel

```
Input: $\omega$, scalar $k$, $P \in E(\mathbb{F}_{2^m})$ of odd order $r$, constant $n$ (e.g., from
    Table 1(b))
Output: $kP$
Compute $P_i = iP$ for $i \in I = \{1, 3, \ldots, 2^{\omega-1} - 1\}$
$Q_0 \leftarrow \mathcal{O}$
Recode: $k' = 2^n k \bmod r$ and obtain rep $\omega\mathrm{NAF}(k')/2^n = \sum_{i=0}^{t} k'_i 2^{i-n}$
Initialize $Q_i \leftarrow \mathcal{O}$ for $i \in I$
{Barrier}
for $i = t$ downto $n$ do
    $Q_0 \leftarrow 2Q_0$
    if $k'_i > 0$ then
        $Q_0 \leftarrow Q_0 + P_{k'_i}$
    else if $k'_i < 0$ then
        $Q_0 \leftarrow Q_0 - P_{-k'_i}$
for $i = n - 1$ downto 0 do
    $P \leftarrow P/2$
    if $k'_i > 0$ then
        $Q_{k'_i} \leftarrow Q_{k'_i} + P$
    else if $k'_i < 0$ then
        $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
{Barrier}
return $Q \leftarrow Q_0 + \sum_{i \in I} i Q_i$
```

3.3 Side-channel-resistant multiplication on random binary curves

Another approach for scalar multiplication offering some resistance to side-channel attacks was proposed by López and Dahab [35] based on the Montgomery laddering technique. This approach requires $6M + 5S$ in $\mathbb{F}_{2^m}$ per iteration independently of the bit pattern in the scalar, and one of these multiplications is by the curve coefficient b. The curve being lately used for benchmarking purposes [7] at the 128-bit security level is an Edwards curve (CURVE2251) corresponding to the Weierstraß curve (1) with $a = 0$ and $b = z^{13}+z^9+z^8+z^7+z^2+z+1$ over $\mathbb{F}_2[z]/(z^{251}+z^7+z^4+z^2+1)$. It is clear that this curve is especially tailored for this method due to the short length of b, reducing the cost of the algorithm to approximately $5.25M + 5S$ per iteration. At the same time, halving-based approaches are non-optimal for this curve due to the penalties introduced by the 4-cofactor [30]. Considering this and to partially satisfy the side-channel resistance offered by a bitsliced implementation such as [6], we restricted the choices of scalar multiplication at this security level to the Montgomery laddering approach.

4 Koblitz elliptic curves

A Koblitz curve $E_a(\mathbb{F}_q)$, also known as an Anomalous Binary Curve [32], is a special case of (1) where $b = 1$ and $a \in \{0, 1\}$. In a binary field, the map taking x to x^2 is an automorphism known as the Frobenius map. Since Koblitz curves are defined over the binary field $\mathbb{F}_2$, the Frobenius map and its inverse naturally extend to automorphisms of the curve denoted τ and τ^{-1}, respectively, where $\tau(x, y) = (x^2, y^2)$. Moreover, $(x^4, y^4)+2(x, y) = \mu(x^2, y^2)$ for every (x, y) on E_a, where $\mu = (-1)^{1-a}$; that is, τ satisfies $\tau^2+2 = \mu\tau$ and we can associate τ with the complex number $\tau = \frac{\mu+\sqrt{-7}}{2}$.

Solinas [41] presents a τ-adic analogue of the usual NAF as follows. Since short representations are desirable, an element $\rho \in \mathbb{Z}[\tau]$ is found with $\rho \equiv k \pmod{\delta}$ of as small norm as possible, where $\delta = (\tau^m - 1)/(\tau - 1)$. Then for the subgroup of interest, $kP = \rho P$ and a width-ω τ-adic NAF ($\omega\tau$NAF) for ρ is obtained in a fashion that parallels the usual ωNAF. As in [41], define $\alpha_i = i \bmod \tau^\omega$ for $i \in \{1, 3, \ldots, 2^{\omega-1} - 1\}$. A $\omega\tau$NAF of a nonzero element ρ is an expression $\rho = \sum_{i=0}^{l-1} u_i \tau^i$ where each $u_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{\omega-1}-1}\}$, $u_{l-1} \neq 0$, and at most one of any consecutive ω coefficients is nonzero. Scalar multiplication kP can be performed with the $\omega\tau$NAF expansion of ρ as

$$u_{l-1}\tau^{l-1}P + \cdots + u_2\tau^2 P + u_1\tau P + u_0 P \qquad (3)$$

with $l - 1$ applications of τ and approximately $l/(\omega + 1)$ additions.

The length of the representation is at most $m + a$, and Solinas presents an efficient technique to find an estimate for ρ, denoted $\rho' = k$ partmod δ with $\rho' \equiv \rho \pmod{\delta}$, having expansion of length at most $m + a + 3$ [9,41]. Under reasonable assumptions, the algorithm will usually produce an estimate giving length at most $m + 1$. For simplicity, we will assume that the recodings obtained have this as an upper bound on length; small adjustments are necessary to process longer representations. Under these assumptions and properties of τ, scalars may be written $k = \sum_{i=0}^{m} u_i \tau^i = \sum_{i=0}^{m} u_i \tau^{-(m-i)}$ since $\tau^{-i} = \tau^{m-i}$ for all i.

4.1 Sequential algorithms for Koblitz curves

Algorithm 5 is a traditional left-to-right τ-and-add method, and expression (3) is computed from left to right, i.e., it starts processing u_{l-1} first, then u_{l-2} until it ends with the coefficient u_0. Step 1 computes $2^{\omega-2}-1$ multiples of the point P, each at a cost of approximately one point addition (see Table 1 for a summary of the computational effort associated to this phase).

Algorithm 5 Left-to-right τ-and-add scalar multiplication

```
Input: $\omega, k \in [1, r-1]$, $P \in E_a(\mathbb{F}_{2^m})$ of order $r$
Output: $kP$
1: Compute $P_u = \alpha_u P$ for $u \in \{1, 3, 5, \ldots, 2^{\omega-1}-1\}$ where $\alpha_i = i \bmod \tau^{\omega}$
2: Compute $\rho = k$ partmod $\delta$ and $\omega\tau\mathrm{NAF}(\rho) = \sum_{i=0}^{l-1} u_i \tau^i$; $Q \leftarrow \mathcal{O}$
3: for $i = l-1$ downto 0 do
4:     $Q \leftarrow \tau Q$
5:     if $u_i = \alpha_j$ then
6:         $Q \leftarrow Q + P_j$
7:     else if $u_i = -\alpha_j$ then
8:         $Q \leftarrow Q - P_j$
9: return $Q$
```

Alternatively, we can process right-to-left as shown in Algorithm 6. The multiple points of precomputation P_u in Algorithm 5 are exchanged for the same number of accumulators Q_u along with postcomputation. The cost of postcomputation is likely more than the precomputation of Algorithm 5; see Table 1 for a summary in the case where postcomputation is with projective additions. However, if the accumulator in Algorithm 5 is in projective coordinates, then Algorithm 6 has a less expensive evaluation phase since τ is applied to points in affine.

Algorithm 6 Right-to-left τ-and-add scalar multiplication

```
Input: $\omega, k \in [1, r-1]$, $P \in E_a(\mathbb{F}_{2^m})$ of order $r$
Output: $kP$
1: $\rho = k$ partmod $\delta$, $\omega\tau\mathrm{NAF}(\rho) = \sum_{i=0}^{l-1} u_i \tau^i$ $Q_u = \mathcal{O}$ for $u \in I = \{1, 3, \ldots, 2^{\omega-1}-1\}$
2: for $i = 0$ to $l-1$ do
3:     if $u_i = \alpha_j$ then
4:         $Q_j \leftarrow Q_j + P$
5:     else if $u_i = -\alpha_j$ then
6:         $Q_j \leftarrow Q_j - P$
7:     $P \leftarrow \tau P$
8: return $Q \leftarrow \sum_{u \in I} \alpha_u Q_u$
```

4.2 Parallel algorithm for Koblitz curves

The basic strategy in our parallel algorithm is to reformulate the scalar multiplication in terms of both the τ and the τ^{-1} operators as $k = \sum_{i=0}^{m} u_i \tau^i = u_0 + u_1\tau^1 + \cdots + u_n\tau^n + u_{n+1}\tau^{-(m-n-1)} + \cdots + u_m = \sum_{i=0}^{n} u_i\tau^i + \sum_{i=n+1}^{m} u_i\tau^{-(m-i)}$ where $0 < n < m$. Algorithm 7 illustrates a parallel approach suitable for two processors. Although similar in structure to Algorithm 4, a significant difference is the shared precomputation rather than the pre and postcomputation required in Algorithm 4.

The scalar representation is given by Solinas [41] and hence has an expected $m/(\omega+1)$ point additions in the evaluation-stage and an extra point addition at the end. There are also approximately m applications of τ or its inverse. If the field representation is such that these operators have similar cost or are sufficiently inexpensive relative to field multiplication, then the evaluation stage can be a factor 2 faster than a corresponding non-parallel algorithm.

Algorithm 7 $\omega\tau$NAF scalar multiplication: parallel

```
Input: $\omega, k \in [1, r-1]$, $P \in E_a(\mathbb{F}_{2^m})$ of order $r$, constant $n$ (e.g., from Table 1(b))
Output: $kP$
1: $\rho \leftarrow k$ partmod $\delta$                  3: $P_u = \alpha_u P$,
2: $\sum_{i=0}^{l-1} u_i\tau^i \leftarrow \omega\tau\mathrm{NAF}(\rho)$          for $u \in \{1, 3, 5, \ldots, 2^{\omega-1}-1\}$
   {Barrier}
4: $Q_0 \leftarrow \mathcal{O}$                          11: $Q_1 \leftarrow \mathcal{O}$
5: for $i = n$ downto 0 do                    12: for $i = n+1$ to $m$ do
6:     $Q_0 \leftarrow \tau Q_0$                      13:     $Q_1 \leftarrow \tau^{-1} Q_1$
7:     if $u_i = \alpha_j$ then                   14:     if $u_i = \alpha_j$ then
8:         $Q_0 \leftarrow Q_0 + P_j$                 15:         $Q_1 \leftarrow Q_1 + P_j$
9:     else if $u_i = -\alpha_j$ then             16:     else if $u_i = -\alpha_j$ then
10:        $Q_0 \leftarrow Q_0 - P_j$                 17:         $Q_1 \leftarrow Q_1 - P_j$
   {Barrier}
18: return $Q \leftarrow Q_0 + Q_1$
```

As discussed earlier, unlike the ordinary width-ω NAF, the τ-adic version requires a relatively expensive calculation to find a short ρ with $\rho \equiv k \pmod{\delta}$. Hence, (a portion of) the precomputation is "free" in the sense that it occurs during scalar recoding. This can encourage the use of a larger window size ω. The essential features exploited by Algorithm 7 are that the scalar can be efficiently represented in terms of the Frobenius map and that the map and its inverse can be efficiently computed, and hence the algorithm adapts to curves defined over small fields.

Algorithm 7 is attractive in the sense that two processors are directly supported without "extra" computations. However, if multiple applications of the "doubling step" are sufficiently inexpensive, then more processors and additional curves can be accommodated in a straightforward fashion without sacrificing the high-level parallelism of Algorithm 7. As an example for Koblitz curves, a variant on Algorithm 7 discards the applications of τ^{-1} (which may be more expensive than τ) and finds $kP = k^1(\tau^j P) + k^0 P = \tau^j(k^1 P) + k^0 P$ for suitable k^i and $j \approx m/2$ with traditional methods to calculate $k^i P$. The application of τ^j is low cost if there is storage for a per-field matrix as it was first discussed in [1].

5 Experimental results

We consider example fields $\mathbb{F}_{2^m}$ for $m \in \{233, 251, 409\}$. These were chosen to address 112-bit and 192-bit security levels, according to the NIST recommendation, and the 251-bit binary Edwards elliptic curve presented in [6]. The field $\mathbb{F}_{2^{233}}$ was also chosen as more likely to expose any overhead penalty in the parallelization compared with larger fields from NIST. Our C library coded all the algorithms using the GNU C 4.6 (GCC) and Intel 12 (ICC) compilers, and the timings were obtained on a 3.326 GHz 32nm Intel Westmere Core i5 660 processor.

Obtaining times useful for comparison across similar systems can be problematic. Intel, for example, introduced "Pentium 4" processors that were fundamentally different than earlier designs with the same name. The common method via time stamp counter (TSC) requires care on recent processors having "turbo" modes that increase the clock (on perhaps 1 of 2 cores) over the nominal clock implicit in TSC, giving an underestimate of actual cycles consumed. Benchmarking guidelines on eBACS [7], for example, recommend disabling such modes, and this is the method followed in this paper.

Timings for field arithmetic as measured in the i5 processor are shown in Table 2. The López–Dahab multiplier described in [2] was implemented as a baseline to quantify the speedup due to the native multiplier. For the most part, timings for GCC and ICC are similar, although López–Dahab multiplication is an exception. The difference in multiplication times between $\mathbb{F}_{2^{233}} = \mathbb{F}_2[z]/(z^{233}+z^{74}+1)$ and $\mathbb{F}_{2^{251}} = \mathbb{F}_2[z]/(z^{251}+z^7+z^4+z^2+1)$ is in reduction. The relatively expensive square root in $\mathbb{F}_{2^{251}}$ is due to the representation chosen; if square roots are of interest, then there are reduction polynomials giving faster square root and similar numbers for other operations. Inversion via exponentiation (Sect. 2) gives I/M similar to that in [2] where a Euclidean algorithm variant was used with similar hardware but without the carry-less multiplier.

Table 4 shows timings obtained for different variants of sequential and parallel scalar multiplication over random binary curves as measured in the i5 processors. We observe that for ωNAF recoding with $\omega = 3, 4$, the halve-and-add algorithm is always faster than its double-and-add counterpart. This performance is a direct consequence of the timings reported in Table 3, where the cost of one point doubling is roughly 5.6 and 4.8 multiplications, whereas the cost of a point halving is of only 3.3 and 2.5 multiplications in the fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{409}}$, respectively. The parallel version that concurrently executes these algorithms in two threads computes one scalar multiplication with a latency that is roughly 37% smaller than that of the halve-and-add algorithm for the curves B-233 and B-409.

Table 5 shows timings obtained for different variants of sequential and parallel scalar multiplication over Koblitz curves as measured in the i5 processor. The bold entries for Koblitz curves identify fastest timings per category (i.e., considering the compiler, curve and the specific value of ω used in the ω NAF recoding). For smaller ω, Algorithm 6 has an edge over Algorithm 5 because τ is applied to points in affine coordinates; this advantage diminishes with increasing ω due to postcomputation cost. "(τ, τ)-and-add" denotes the parallel variant described in Sect. 4.2. There is a storage penalty for a linear map, but applications of τ^{-1} are eliminated (of interest when τ is significantly less expensive). Given the modest cost of the multi-squaring operation (with an equivalent cost of less than 1.44 field multiplications, see Table 2), the (τ, τ)-and-add parallel variant is usually faster than Algorithm 7. When using $\omega = 5$, the parallel (τ, τ)-and-add algorithm computes one scalar multiplication with a latency that is roughly 35.6 and 40.5% smaller than that of the best sequential algorithm for the curves K-233 and K-409, respectively.

Table 4 Timings in 10^3 clock cycles for random curve scalar multiplication in the unknown-point scenario measured on a Westmere processor

ω	Scalar mult random curves	B-233		B-409	
		GCC	ICC	GCC	ICC
3	Double-and-add	240	238	984	989
	Halve-and-add	196	192	755	756
	(Dbl, Halve)-and-add	122	118	465	466
4	Double-and-add	231	229	941	944
	Halve-and-add	188	182	706	705
	(Dbl, Halve)-and-add	122	116	444	445
	Side-channel resistant	CURVE2251			
	scalar multiplication	GCC		ICC	
	Montgomery laddering	296		282	

Table 5 Timings in 10^3 clock cycles for Koblitz curve scalar multiplication in the unknown-point scenario measured on a Westmere processor

ω	Scalar mult Koblitz curves	K-233		K-409	
		GCC	ICC	GCC	ICC
3	Alg. 5	111	110	413	416
	Alg. 6	98	**98**	**381**	389
	(τ, τ)-and-add	**73**	74	**248**	248
	Alg. 7	80	78	253	248
4	Alg. 5	97	95	353	355
	Alg. 6	90	**89**	**332**	339
	(τ, τ)-and-add	68	**65**	216	214
	Alg. 7	73	69	218	**214**
5	Alg. 5	92	**90**	326	328
	Alg. 6	95	93	**321**	332
	(τ, τ)-and-add	63	**58**	197	**191**
	Alg. 7	68	63	197	194

Table 6 Comparison with hardware accelerators for elliptic curve scalar multiplication

	Curve	Security (bits)	Platform and area	Calc. time (µs)	Throughput (Mbps)
Järvinen and Skyttä [26]	K-163	80	Stratix II, 23346 ALMs	28.95	5.63
	K-163	80	Stratix II, 13472 ALMs	20.28	8.04
Järvinen and Skyttä [27]	K-163	80	Stratix II, 26148 ALMs	4.91	33.20
Lutz and Hasan [37]	K-163	80	Virtex E, 10017 LUTs	75.00	2.17
Ahmadi et al.[a] [1]	K-233	112	Virtex 2, 15916 Slices	7.22	32.27
This work	K-233	112	Intel i5 660 @3.326GHz[b], $^{-}$	17.50	13.32
	curve2251	128	Intel i5 660 @3.326GHz[c], $^{-}$	79.40	3.16
	K-409	192	Intel i5 660 @3.326GHz[b], $^{-}$	57.37	7.13

[a] Time and area costs of $\omega\tau$ NAF expansion were not included
[b] Two-core Implementation
[c] Single-core Implementation

Table 7 Timings in clock cycles for field arithmetic operations on a Sandy Bridge processor

Base field operation	$\mathbb{F}_{2^{233}}$			$\mathbb{F}_{2^{251}}$			$\mathbb{F}_{2^{409}}$		
	GCC	ICC	op/M	GCC	ICC	op/M	GCC	ICC	op/M
Multiplication	100	100	1.00	130	126	1.00	270	273	1.00
López–Dahab Mult.	210	320	3.20	276	398	3.16	491	660	2.42
Square root	63	56	0.56	127	131	1.04	52	49	0.18
Squaring	22	24	0.24	57	47	0.37	35	34	0.12
Half trace	142	129	1.29	156	148	1.17	254	242	0.88
Multi-Squaring	133	112	1.12	141	120	0.95	419	392	1.43
Inversion	2,215	2,110	21.10	3,202	3,058	24.27	7,256	7,858	28.78
3-τNAF	6,933	9,572	95.72	–	–	–	16,353	20,272	74.26
4-τNAF	6,550	9,266	92.66	–	–	–	15,722	19,758	72.37
5-τNAF	6,351	9,033	90.33	–	–	–	15,221	19,363	70.93
3-NAF	2,859	3,245	32.45	–	–	–	9,151	9,165	33.57
4-NAF	2,402	2,714	27.14	–	–	–	7,793	7,803	25.58
Recoding (halving)	1,454	1,465	14.65	–	–	–	2,765	2,954	10.82
Recoding (parallel)	951	1,011	10.11	–	–	–	1,839	1,987	7.28

op/M denotes ratio to multiplication obtained from ICC

Per-field storage and coding techniques compute half-trace at cost comparable to field multiplication, and methods based on halving continue to be fastest for suitable random curves. However, the hardware multiplier and squaring (via shuffling) give a factor 2 advantage to Koblitz curves in the examples from NIST. This is larger than in [16,21], where a 32-bit processor in the same general family as the i5 has half-trace at approximately half the cost of a field multiplication for B-233 and a factor 1.7 advantage to K-163 over B-163 (and the factor would have been smaller for K-233 and B-233). It is worth remarking that the parallel scalar multiplications versions shown in Tables 5 and 10 look best for bigger curves and larger ω.

Somewhat surprisingly, our software implementations outperform several hardware accelerators previously reported in the open literature (see Table 6). In the case of relatively low security levels, hardware implementations of elliptic curve scalar multiplication remain faster. For example, the computation of point multiplication on the NIST K-163 curve performed on a Stratix II as reported in [27] is roughly 3.6 times faster than our two-core point multiplication implementation on NIST K-233. Depending on the application, this speedup may justify the usage of larger FPGAs which are now available in hybrid computers.

5.1 First look at Sandy Bridge implementation issues

In this section we present the performance achieved by our C library when implemented on a 3.4 GHz 32 nm Intel *Sandy Bridge* Core i7 2600K processor. Corresponding timings for field and elliptic curve arithmetic and sequential and parallel scalar multiplication over random binary curves and Koblitz curves as measured in the Sandy Bridge processor are shown in Tables 7, 8, 9 and 10, respectively.

Sandy Bridge possesses 256-bit registers with new addressing modes [33]. All SSE variants from the Pentium III to the i5 have the limitation that wide-register operations place the output in one of the input operands. The AVX instruction set in Sandy Bridge permits a target register for the result of many operations, thereby reducing the number of explicit

Table 8 Timings in clock cycles for curve arithmetic operations on a Sandy Bridge processor

Elliptic curve operations	B-233			B-409		
	GCC	ICC	op/M	GCC	ICC	op/M
Doubling (LD)	540	540	5.40	1,283	1,291	4.73
Addition (KIM Mixed)	953	932	9.32	2,362	2,352	8.62
Addition (LD Mixed)	959	953	9.53	2,395	2,413	8.84
Addition (LD General)	1,530	1,517	15.17	3,822	3,848	14.10
Halving	391	387	3.87	755	707	2.58

op/M denotes ratio to multiplication obtained from ICC

Table 9 Timings in 10^3 clock cycles for random curve scalar multiplication in the unknown-point scenario measured on a Sandy Bridge processor

ω	Scalar mult	B-233		B-409	
	random curves	GCC	ICC	GCC	ICC
	Double-and-add	189	185	771	768
3	Halve-and-add	165	163	596	610
	(Dbl, Halve)-and-add	102	102	364	373
	Double-and-add	182	178	738	738
4	Halve-and-add	160	157	557	574
	(Dbl, Halve)-and-add	102	100	349	358
	Side-channel resistant	CURVE2251			
	scalar multiplication	GCC		ICC	
	Montgomery laddering	245		225	

Table 10 Timings in 10^3 clock cycles for Koblitz curve scalar multiplication in the unknown-point scenario measured on a Sandy Bridge processor

ω	Scalar mult Koblitz curves	K-233		K-409	
		GCC	ICC	GCC	ICC
	Alg. 5	83.8	84.0	325.9	325.5
3	Alg. 6	75.0	**74.9**	**299.8**	301.5
	(τ, τ)-and-add	59.5	**58.6**	195.0	**191.6**
	Alg. 7	64.6	63.3	196.3	194.5
	Alg. 5	72.9	73.1	277.0	277.1
4	Alg. 6	69.3	**67.8**	**261.5**	262.4
	(τ, τ)-and-add	53.3	**50.8**	167.5	**165.2**
	Alg. 7	57.9	55.7	168.2	166.3
	Alg. 5	**69.4**	69.9	**255.6**	256.2
5	Alg. 6	73.0	72.4	255.7	257.1
	(τ, τ)-and-add	48.2	**46.5**	154.1	**148.8**
	Alg. 7	54.9	51.0	154.0	150.3

move instructions required for operations of interest here and allowing better register allocation due to higher free register availability.

The initial AVX offerings target floating-point operations, and only a portion of the code here benefits from the increased width to 256 bits. The code for half-trace and multi-squaring, for example, can exploit the 256-bit registers and explains a significant portion of the 28–42% improvement over times on the i5 for $\mathbb{F}_{2^{251}}$. The improvement for field multiplication with the carryless multiplier is 20%, due to improved register allocation and latency of the instruction. The method of López–Dahab benefits from AVX addressing, but was implemented with 128-bit operations due to the lack of a suitable shift in the 256-bit registers. We remark that the times for López–Dahab multiplication with the Intel compiler require more investigation to understand the poor performance relative to the GNU compiler.

6 Conclusion and future work

In this work we achieve the fastest timings reported in the open literature for software computation of scalar multiplication in NIST and Edwards binary elliptic curves defined at the 112-bit, 128-bit and 192-bit security levels. The two-core implementation of scalar multiplication over the NIST K-233 elliptic curve on the Westmere and Sandy Bridge processors can be computed in less than 17.5 and 13.8 μs, respectively. These results are not only much faster than previous software implementations of that curve, but are also quite competitive with the computation time achieved by state-of-the-art hardware accelerators working on similar or smaller curves [1,25]. For both compatibility and dissemination purposes we are currently in the process of benchmarking our cryptographic library using the publicly available SUPERCOP suite [7].

These fast timings were obtained through the usage of the native carry-less multiplier available in the newest Intel processors. At the same time, we strive to use the best algorithmic techniques and the most efficient elliptic curve and finite field arithmetic formulae. Further, we proposed effective parallel formulations of scalar multiplication algorithms suitable for deployment in multi-core platforms.

The curves over binary fields permit relatively elegant parallelization with low synchronization cost, mainly due to the efficient halving or τ^{-1} operations. Parallelizing at lower levels in the arithmetic would be desirable, especially for curves over prime fields. Grabher et al. [17] apply parallelization for extension field multiplication, but times for a base field multiplication in a 256-bit prime field are relatively slow compared with Beuchat et al. [8]. On the other hand, a strategy that applies to all curves performs point doubles in one thread and point additions in another. The doubling thread stores intermediate values corresponding to nonzero digits of the NAF; the addition thread processes these points as they become available. Experimentally, synchronization cost is low, but so is the expected acceleration. Against the fastest times in Longa and Gebotys [34] for a curve over a 256-bit prime field, the technique would offer roughly 17% improvement, a disappointing return on processor investment.

To the best of our knowledge, the current scalar multiplication speed record for a 128-bit security level single-core software implementation is held by the work reported in [22], where one scalar multiplication using a 4-Dimensional GLV method on GLS elliptic curves with Jacobian coordinates is computed in just 122 thousand cycles on a 2.7GHz Intel Core i7-2620M processor.

On the other hand, using the methods described in this paper, scalar multiplication on the NIST K-233 curve was computed in 68 thousand cycles when implemented on a Sandy Bridge processor (see Table 10). From this result, we believe that by adjusting our library for handling field and curve arithmetic on NIST K-283, a curve that enjoys 128-bit security level, it should be possible to compute a single scalar multiplication in a timing competitive with the best implementation over a prime field.

We are currently working on computing scalar multiplication on the NIST K-283 curve. After the completion of this project we expect to be able to answer positively the question of whether the most efficient elliptic curves defined over binary fields are faster than their prime field counterparts when implemented in state-of-the-technology single-core front end processors at the 128-bit security level.

The new native support for binary field multiplication allowed our implementation to improve by 10–28% on the previous speed record for side-channel-resistant scalar multiplication in random elliptic curves. It is hard to predict what will be the superior strategy between a conventional non-bitsliced or a bitsliced implementation on future revisions of the target platform: the latency of the carry-less multiplier instruction has clear room for improvement, while the new AVX instruction set has 256-bit registers. An issue with the current Sandy Bridge version of AVX is that `xor` throughput for operations with register operands was decreased significantly from 3 operations per cycle in SSE to 1 operation per cycle in AVX. The resulting performance of a bitsliced implementation will ultimately rely on the amount of work which can be scheduled to be done mostly in registers.

Acknowledgments We wish to thank the University of Waterloo and especially Professor Alfred Menezes for useful discussions related to this work during a visit by three of the authors, where the idea of this project was discussed, planned and a portion of the development phase was done. Diego F. Aranha and Julio López thank CNPq, CAPES and FAPESP for financial support. Jonathan Taverne, Armando Faz-Hernández and Francisco Rodríguez-Henríquez thank CONACyT grant 60240 and UCMEXUS for financial support.

References

1. Ahmadi, O., Hankerson, D., Rodríguez-Henríquez, F.: Parallel formulations of scalar multiplication on Koblitz curves. J. UCS **14**(3), 481–504 (2008)
2. Aranha, D.F., López, J., Hankerson, D.: Efficient software implementation of binary field arithmetic using vector instruction sets. In: Abdalla, M., Barreto, P.S.L.M. (eds.) The First International Conference on Cryptology and Information Security (LATINCRYPT 2010). Lecture Notes in Computer Science, vol. 6212, pp. 144–161 (2010)
3. Avanzi, R.M.: Another look at square roots (and other less common operations) in fields of even characteristic. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) 14th International Workshop on Selected Areas in Cryptography (SAC 2007). Lecture Notes in Computer Science, vol. 4876, pp. 138–154. Springer (2007)
4. Bellare, M. (ed.): Advances in Cryptology—CRYPTO 2000. Lecture Notes in Computer Science, vol. 1880. Springer (2000)
5. Bernstein, D., Lange, T.: Analysis and optimization of elliptic-curve single-scalar multiplication. In: Proceedings 8th International Conference on Finite Fields and Applications (Fq8), vol. 461, pp. 1–20. AMS (2008)
6. Bernstein, D.J.: Batch Binary Edwards. In: Halevi, S. (ed.) Advances in Cryptology—CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 317–336. Springer (2009)
7. Bernstein, D.J., Lange, T. (eds.) eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to. Accessed 25 Aug 2011
8. Beuchat, J.-L., Díaz, J., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing-Based Cryptography—Pairing 2010. Lecture Notes in Computer Science, vol. 6487, pp. 21–39 (2010)
9. Blake, I.F., Murty, V.K., Xu, G.: A note on window τ-NAF algorithm. Inf. Process. Lett. **95**(5), 496–502 (2005)
10. Bodrato, M.: Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In: Carlet, C., Sunar, B. (eds.) Arithmetic of Finite Fields (WAIFI 2007). Lecture Notes in Computer Science, vol. 4547, pp. 116–133. Springer (2007)
11. Bos, J.W., Kleinjung, T., Niederhagen, R., Schwabe, P.: ECC2K-130 on Cell CPUs. In: Bernstein, D.J., Lange, T. (eds.) 3rd International Conference on Cryptology in Africa (AFRICACRYPT 2010). Lecture Notes in Computer Science, vol. 6055, pp. 225–242. Springer (2010)
12. Comba, P.G.: Exponentiation cryptosystems on the IBM PC. IBM Syst. J. **29**(4), 526–538 (1990)
13. Dahmen, E., Okeya, K., Schepers, D.: Affine precomputation with sole inversion in elliptic curve cryptography. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) Information Security and Privacy (ACISP 2007). Lecture Notes in Computer Science, vol. 4586, pp. 245–258. Springer (2007)
14. Firasta, N., Buxton, M., Jinbo, P., Nasri, K., Kuo, S.: Intel AVX: new frontiers in performance improvement and energy efficiency. White paper. http://software.intel.com/
15. Fog, A.: Instruction tables: list of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. http://www.agner.org/optimize/instruction_tables.pdf. Accessed 01 Mar 2011
16. Fong, K., Hankerson, D., López, J., Menezes, A.: Field inversion and point halving revisited. IEEE Trans. Comput. **53**(8), 1047–1059 (2004)
17. Grabher, P., Großschädl, J., Page, D.: On software parallel implementation of cryptographic pairings. Cryptology ePrint Archive, Report 2008/205. http://eprint.iacr.org/ (2008)

18. Guajardo, J., Paar, C.: Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. Des. Codes Cryptogr. **25**(2), 207–216 (2002)
19. Gueron, S.: Intel Advanced Encryption Standard (AES) Instructions Set. White paper. http://software.intel.com/
20. Gueron, S., Kounavis, M.E.: Carry-less multiplication and its usage for computing the GCM mode. White paper. http://software.intel.com/
21. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Secaucus (2004)
22. Hu, Z., Longa P., Xu, M.: Implementing 4-dimensional GLV method on GLS elliptic curves with j-invariant 0. Des. Codes Cryptogr. (to appear)
23. Intel.: Intel SSE4 Programming Reference. Technical Report. http://software.intel.com/
24. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF (2^m) using normal bases. Inf. Comput. **78**(3), 171–177 (1988)
25. Järvinen, K., Optimized FPGA-based elliptic curve cryptography processor for high-speed applications. Integr. VLSI J. (to appear)
26. Järvinen, K.U., Skyttä, J.: On parallelization of high-speed processors for elliptic curve cryptography. IEEE Trans. VLSI Syst. **16**(9), 1162–1175 (2008)
27. Järvinen, K.U., Skyttä, J.: Fast point multiplication on Koblitz curves: Parallelization method and implementations. Microprocess. Microsyst. Embedded Hardware Des. **33**(2), 106–116 (2009)
28. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Doklady Akad. Nauk SSSR **145**, 293–294 (1962). Translation in Physics-Doklady **7**, 595–596 (1963)
29. Kim, K.H., Kim, S.I.: A new method for speeding up arithmetic on elliptic curves over binary fields. Cryptology ePrint Archive, Report 2007/181. http://eprint.iacr.org/ (2007)
30. King, B. Rubin, B.: Improvements to the point halving algorithm. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) 9th Australasian Conference on Information Security and Privacy (ACISP 2004). Lecture Notes in Computer Science, vol. 3108, pp. 262–276. Springer (2004)
31. Knudsen, E.: Elliptic scalar multiplication using point halving. In: Lam, K., Okamoto, E. (eds.) Advances in Cryptology—ASIACRYPT '99. Lecture Notes in Computer Science, vol. 1716, pp. 135–149. Springer (1999)
32. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) Advances in Cryptology—CRYPTO '91. Lecture Notes in Computer Science, vol. 576, pp. 279–287. Springer (1992)
33. Lomont, C.: Introduction to Intel advanced vector extensions. Intel Software Network. http://software.intel.com/file/37205 (2011)
34. Longa, P., Gebotys, C.H.: Efficient techniques for high-speed elliptic curve cryptography. In: Mangard, S., Standaert, F.-X. (eds.) Cryptographic Hardware and Embedded Systems (CHES 2010). Lecture Notes in Computer Science, vol. 6225, pp. 80–94. Springer (2010)
35. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m) without precomputation. In: Koç, Ç.K., Paar, C. (eds.) First International Workshop on Cryptographic Hardware and Embedded Systems (CHES 99). Lecture Notes in Computer Science, vol. 1717, pp. 316–327. Springer (1999)
36. López, J., Dahab, R.: High-speed software multiplication in GF(2^m). In: Roy, B.K., Okamoto, E. (eds.) 1st International Conference in Cryptology in India (INDOCRYPT 2000). Lecture Notes in Computer Science, vol. 1977, pp. 203–212. Springer (2000)
37. Lutz, J., Hasan, M.A.: High performance FPGA based elliptic curve cryptographic co-processor. In: International Conference on Information Technology: Coding and Computing (ITCC'04), vol. 2, pp. 486–492. IEEE Computer Society (2004)
38. Montgomery, P.L.: Five, six, and seven-term Karatsuba-like formulae. IEEE Trans. Comput. **54**(3), 362–369 (2005)
39. National Institute of Standards and Technology (NIST).: Recommended Elliptic Curves for Federal Government Use. NIST Special Publication. http://csrc.nist.gov/csrc/fedstandards.html. Accessed July 1999
40. Schroeppel, R.: Elliptic curves: Twice as fast! Presentation at the CRYPTO 2000 [4] Rump Session (2000)
41. Solinas, J.A.: Efficient arithmetic on Koblitz curves. Des. Codes Cryptogr. **19**(2-3), 195–249 (2000)
42. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Software implementation of binary elliptic curves: impact of the carry-less multiplier on scalar multiplication. In: International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2011). Lecture Notes in Computer Science, vol. 6917, Springer, New York (2011)
43. Wall, D.W.: Limits of instruction-level parallelism. In: 4th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS 91), pp. 176–188. ACM, New York (1991)
44. Wulf, W.A., McKee, S.A.: Hitting the memory wall: implications of the obvious. SIGARCH Comput. Architect. News **23**(1), 20–24 (1995)