Batch binary Edwards

Daniel J. Bernstein [⋆]

Department of Computer Science (MC 152)
The University of Illinois at Chicago
Chicago, IL 60607–7053
djb@cr.yp.to

Abstract. This paper sets new software speed records for high-security Diffie-Hellman computations, specifically 251-bit elliptic-curve variable-base-point scalar multiplication. In one second of computation on a $200 Core 2 Quad Q6600 CPU, this paper's software performs 30000 251-bit scalar multiplications on the binary Edwards curve $d(x + x^2 + y + y^2) = (x + x^2)(y + y^2)$ over the field $\mathbf{F}_2[t]/(t^{251} + t^7 + t^4 + t^2 + 1)$ where $d = t^{57} + t^{54} + t^{44} + 1$. The paper's field-arithmetic techniques can be applied in much more generality but have a particularly efficient interaction with the completeness of addition formulas for binary Edwards curves.

Keywords. Scalar multiplication, Diffie–Hellman, batch throughput, vectorization, Karatsuba, Toom, elliptic curves, binary Edwards curves, differential addition, complete addition formulas

1 Introduction

Which curves should one choose for elliptic-curve cryptography?

The first and most fundamental choice is between curves defined over binary (i.e., characteristic-2) finite fields and curves defined over non-binary fields. For example:

- NIST's standard K-283 curve is the subfield curve $y^2 + xy = x^3 + 1$ over the field $\mathbf{F}_2[t]/(t^{283} + t^{12} + t^7 + t^5 + 1)$. NIST's standard B-283 curve is a particular non-subfield curve over the same binary field.
- NIST's standard P-256 curve is a particular curve over the prime field $\mathbf{Z}/(2^{256} - 2^{224} + 2^{192} + 2^{96} - 1)$.

Multiplication in the polynomial ring $\mathbf{F}_2[t]$ is, at first glance, just like multiplication in $\mathbf{Z}$ but skips all the carries. Furthermore, squaring in $\mathbf{F}_2[t]$ is simply a relabeling of exponents. One might therefore guess that curves over binary fields are considerably faster than curves over large-characteristic fields.

However, the software speed records for elliptic-curve cryptography are—and for many years have been—held by large-characteristic fields. The fastest

[⋆] Permanent ID of this document: 4d7766189e82c1381774dc840d05267b. Date of this document: 2009.06.03. This work was supported by the National Science Foundation under grant ITR–0716498.

Diffie–Hellman speeds (i.e., speeds for variable-base-point scalar multiplication $n, P \mapsto nP$) reported in ECRYPT's publicly verifiable benchmarks [15] on a single core of an Intel Core 2 Quad Q6600 (6fb, `utrecht`) are

- 321012 cycles for field size $(2^{127} - 1)^2$ (using software from Galbraith, Lin, and Scott, combining Edwards curves with the idea of [33]),
- 386739 cycles for field size $2^{255} - 19$ (using software from Gaudry and Thomé announced in [35]), and
- 855036 cycles for field size 2^{251} (also using software from [35]).

Similar comments apply to older processors: for example, Fong, Hankerson, López, and Menezes in [30, Table 6] report 1720000 cycles on a Pentium III for field size 2^{233}, while [12] reports 832457 cycles on a Pentium III for field size $2^{255} - 19$. Subfield curves provide some speedups in the binary case—for example, Hankerson et al. in [38, Table 7] report 1740000 cycles on a Pentium II for field size 2^{283}, and Hankerson, Karabina, and Menezes in [39, Table 5] report 758000 cycles on a Xeon 5460 (similar to a Core 2 Quad) for field size 2^{254}—but binary fields seem to have no hope of catching up to large-characteristic fields.

Why are large-characteristic fields so much faster than binary fields? The conventional explanation is that today's popular CPUs include fast "integer-multiplication" (and "floating-point multiplication") instructions that multiply medium-size elements of $\mathbf{Z}$, but do not include instructions to multiply medium-size elements of $\mathbf{F}_2[t]$. Of course, one can multiply in $\mathbf{F}_2[t]$ by combining simpler CPU instructions, but the multiplication instructions for $\mathbf{Z}$ are much faster, outweighing any possible advantages of characteristic 2. This effect has been intensified by the transition from 32-bit processors to 64-bit processors: 64-bit multipliers are even more powerful than 32-bit multipliers.

Why have CPU designers decided to include a circuit for multiplication in $\mathbf{Z}$ and not a smaller, faster circuit for multiplication in $\mathbf{F}_2[t]$? The conventional explanation is as follows. Most of the computer users who care about CPU speed are measuring performance of weather simulation, movie decompression, video games, etc. These applications rely heavily on multiplication in $\mathbf{Z}$, rewarding CPUs that include integer multipliers, floating-point multipliers, etc. The same applications make very little use of multiplication in $\mathbf{F}_2[t]$.

New speed records. This paper introduces new software named `BBE251` for scalar multiplication on a high-security binary elliptic curve, specifically the binary Edwards curve $d(x + x^2 + y + y^2) = (x + x^2)(y + y^2)$ over k, where $k = \mathbf{F}_{2^{251}} = \mathbf{F}_2[t]/(t^{251} + t^7 + t^4 + t^2 + 1)$ and $d = t^{57} + t^{54} + t^{44} + 1 \in k$. This curve has group order $4 \cdot$ prime and twist order $2 \cdot$ prime, and it satisfies all the usual elliptic-curve security criteria; see Section 3.

`BBE251` is so fast that it sets new speed records not just for binary elliptic curves, but for *all* elliptic curves. For example, a benchmark of a batch of 1024 independent scalar multiplications took 321866514 cycles on a single core of a Core 2 Quad Q6600 (6fb)—a cost of just 314323 Core 2 cycles per scalar multiplication, improving upon all previous results. The Sage computer-algebra system [63] was used to check a random sampling of outputs.

Readers and potential users are cautioned that BBE251 does *not* compute one scalar multiplication in 314323 cycles. The software is given a *batch* of curve points $P_1, P_2, P_3, \ldots$ and an equal-length batch of integers $n_1, n_2, n_3, \ldots$; it produces a batch of multiples $n_1P_1, n_2P_2, n_3P_3, \ldots$. The speed of BBE251 does not rely on any relationships between the inputs; in fact, for fixed-size batches, the software takes constant time, independent of the inputs. BBE251 nevertheless provides speed benefits from handling (n_1, P_1) together with (n_2, P_2) and (n_3, P_3) and so on. BBE251 is faster than the software in [35] for the same field size once the batch size is above 50; for large batches it is more than twice as fast.

Real-world servers bottlenecked by typical elliptic-curve computations can gain speed by collecting the computations into batches, switching to the curve introduced in this paper, and switching to the software introduced in this paper. The batching increases latency by several milliseconds, but in most applications this is not a problem, whereas raw throughput is often critical. BBE251 completes 1048576 scalar multiplications in just 35 seconds using all four cores of a single 2.4GHz Core 2 Quad Q6600 CPU; interference among the cores is negligible. This is not the fastest single-chip scalar-multiplication measurement ever reported in the literature—Güneysu and Paar in [36] reported "more than 37000 point multiplications per second"—but a closer look shows that [36] achieved 37000 224-bit scalar multiplications per second on a \$1000 Xilinx Virtex-4 SX55 containing hundreds of multipliers, while this paper achieves 30000 251-bit scalar multiplications per second on a \$200 Core 2 Quad Q6600.

BBE251 provides several benefits beyond speed. It avoids all data-dependent array indices, all data-dependent branches, etc., and is therefore immune to cache-timing attacks, branch-prediction attacks, etc.; the same security feature was already present in state-of-the-art software for large-characteristic elliptic curves (see, e.g., [12]) but is hard to find for binary curves. BBE251 has been posted (http://binary.cr.yp.to) to allow public verification of its accuracy and speed, and has been placed into the public domain to maximize reusability.

How these speeds were achieved: low level. Schoolbook multiplication of two 251-bit polynomials in $\mathbf{F}_2[t]$ takes 125501 bit operations: specifically, $251^2 = 63001$ bit multiplications (ANDs) and $250^2 = 62500$ bit additions (XORs). BBE251 instead uses several layers of Karatsuba and Toom recursions, including some new refinements, reducing the number of bit operations to 33096, approximately 3.79× smaller than 125501. See Section 2 for details.

The conventional wisdom is that bit-operation counts are a poor predictor of software performance, for two critical reasons:

- CPUs handle multiple bits at once. For example, a 32-bit xor is a single instruction, just as fast as a 16-bit xor or an 8-bit xor. It therefore makes no sense to split a 32×32-bit problem into three 16×16-bit problems or to split a 16×16-bit problem into three 8×8-bit problems.
- $\mathbf{F}_2[t]$-multiplication software is bottlenecked by the cost of shifting bits within words, not by the cost of performing arithmetic on bits. For example, extracting two 4-bit pieces from an 8-bit input is not free; it costs two or three instructions, depending on the CPU.

It is widely appreciated that fast multiplication techniques save time in software *for sufficiently large inputs*, but the speedups are generally believed to be rather small at cryptographic sizes—certainly not a 3.79× speedup for 251-bit multiplication. Hankerson, Hernandez, and Menezes in [38, Section 3] say that Karatsuba is "competitive" for 233 bits and for 283 bits but does not actually save time. Bailey and Paar in [10] say that fast polynomial multiplication "yields a 10% speedup in the overall scalar multiplication time" for a particular curve. Brent, Gaudry, Thomé, and Zimmermann in [22] report speedups from several layers of Karatsuba and Toom recursions, but only beyond cryptographic sizes.

In `BBE251`, a 16-bit xor *is* faster than a 32-bit xor, because two 16-bit inputs are packed into the same space as a 32-bit input and handled in parallel. Shift costs are trivially eliminated by the standard technique of "bitslicing": w separate b-bit inputs $i_0 = (i_{0,0}, i_{0,1}, \ldots, i_{0,b-1})$, $i_1 = (i_{1,0}, i_{1,1}, \ldots, i_{1,b-1})$, $\ldots$ are batched, transposed into b separate w-bit vectors $(i_{0,0}, i_{1,0}, \ldots)$, $(i_{0,1}, i_{1,1}, \ldots)$, $\ldots$, $(i_{0,b-1}, i_{1,b-1}, \ldots)$, and then handled without shifts until the end of the computation. These vectors do not fit simultaneously into CPU registers, but `BBE251` arranges operations so that most loads and stores are overlapped with computation.

Bitslicing has been used in cryptography before. Two record-setting examples are Biham's implementation [18] of DES, a standard hardware-friendly bit-oriented cipher that had previously been viewed as very slow in software, and the Matsui–Nakajima implementation [50] of AES, taking just 9.2 cycles per byte on a Core 2. Aoki, Hoshino, and Kobayashi in [9] pointed out that bitsliced field arithmetic saves time for binary elliptic curves—but they were still unable to compete with non-binary elliptic curves. The Pentium III speeds reported in [9, Table 3] for a subfield curve over a field of size 2^{163} are not as fast as the Pentium II speeds reported the same year by Aoki et al. in [8, Table 4] for a curve over a larger non-binary field. The fast multiplication circuits built into current CPUs perform a huge number of bit operations per cycle and are generally perceived as indispensable tools for fast public-key cryptography; it is surprising that bitsliced field arithmetic can set new Diffie–Hellman speed records, outperforming the multiplication circuits.

How these speeds were achieved: high level. There is a limitation to the power of bitslicing, at least in the pure form used in this paper: bitslicing requires all computations to be expressed as straight-line sequences of bit operations. Straight-line computations do not contain data-dependent branches (e.g., "if $P = Q$ then . . . ") or data-dependent array indices (e.g., "load $x[i]$, where i is the next bit of the scalar"). Computations that include data-dependent array indices can be simulated by straight-line computations that perform arithmetic on all array elements, and computations that include data-dependent branches can be simulated in an analogous way (recall that the "program counter" is just another array index), but these simulations are slow.

Fortunately, a recent line of research has shown how to carry out some elliptic-curve computations without data-dependent array indices, without data-dependent branches, and without serious loss of speed. These techniques have been advertised as efficiently protecting against various types of software side-channel attacks, such as cache-timing attacks, but the same techniques are also useful for efficient bitslicing. Specifically:

- Single-scalar multiplication, odd characteristic: [12, Theorem 2.1] says that the differential-addition formulas from [52] compute $X_0(nP)$ from $X_0(P)$ on any Montgomery curve having a unique 2-torsion point. Here $X_0(P)$ means x if $P = (x, y)$ and 0 if $P = \infty$. See [13, Section 5] for discussion of more general Montgomery curves.
- Arbitrary group operations, odd characteristic: [14, Theorem 3.3] says that various addition formulas are complete—i.e., have no exceptional cases—for any Edwards curve $x^2+y^2 = 1+dx^2y^2$ having non-square d. Complete addition formulas allow complete single-scalar multiplication, complete double-scalar multiplication, etc.
- Arbitrary group operations, characteristic 2: [16, Theorem 4.1] says that various addition formulas in [16] are complete for any binary Edwards curve $d_1(x + y) + d_2(x^2 + y^2) = (x + x^2)(y + y^2)$ having d_2 of trace 1.

The complete differential-addition formulas in [16, Section 7] are reviewed in Section 3 of this paper and are used in `BBE251`. Other approaches, such as mixing bitsliced computation with non-bitsliced computation, do not appear to provide noticeable benefits for the speed of elliptic-curve scalar multiplication and are not discussed in this paper.

The unexpected speed of bitsliced binary-field multiplication can be viewed as motivation to consider bitslicing for a wide variety of higher-level cryptographic computations and for many other problems. This paper's results on binary-elliptic-curve scalar multiplication are one step along this path. The reader is cautioned, however, that bitslicing will provide smaller benefits for computations that rely more heavily on random access to memory.

What about `PCLMULQDQ`? Some CPU designers have begun to realize the potential importance of $\mathbf{F}_2[t]$ for cryptographic applications. Intel announced in April 2008 that its processors will eventually include a "carry-less multiplication" instruction named `PCLMULQDQ`. Intel's white paper [40] says that "carry-less multiplication"—i.e., multiplication in $\mathbf{F}_2[t]$—is particularly important for GCM, a standard for secret-key authenticated encryption relying heavily on multiplications in the binary field $\mathbf{F}_{2^{128}}$; and that "accelerating carry-less multiplication can significantly contribute to achieving high speed secure computing and communication." Gueron and Kounavis estimate in [37] that scalar multiplication on the NIST B-233 elliptic curve would take 220000 cycles using a 9-cycle instruction for "carry-less multiplication," or 70000 cycles using a 3-cycle instruction for "carry-less multiplication." NIST B-233 has a somewhat lower security level than the 251-bit curve used in this paper.

Intel also announced in April 2008 that its processors will, starting in 2010, include 256-bit vector instructions. The initial instruction set will not include many integer instructions but *will* include 256-bit logical operations such as `VXORPS ymm` and `VANDPS ymm`; see [41].

It is clear that PCLMULQDQ-based software will easily outperform most software for binary-elliptic-curve computations. However, it is not at all clear that PCLMULQDQ will outperform a 256-bit-vector implementation of the techniques introduced in this paper. It is not even clear that PCLMULQDQ will outperform the specific BBE251 software described in this paper!

In any case, one can reasonably speculate that continued improvements in binary-elliptic-curve performance will encourage cryptographic users to shift to binary elliptic curves, increasing the importance of techniques to accelerate computations on those curves. Presumably **Z**-biased CPUs will continue being sold for years and will continue being used for years after that; the techniques in this paper will clearly remain important for **Z**-biased CPUs whether or not they are helpful for PCLMULQDQ CPUs.

2 Polynomial multiplication

This section explains how to multiply 251-bit polynomials using 33096 bit operations. More generally, this section presents small explicit upper bounds on $M(n)$, the minimum number of bit operations needed to multiply an n-bit polynomial $f_0 + f_1 t + \cdots + f_{n-1} t^{n-1}$ by another n-bit polynomial $g_0 + g_1 t + \cdots + g_{n-1} t^{n-1}$.

Inputs and outputs are represented in the standard form: $f_0 + f_1 t + \cdots + f_{n-1} t^{n-1}$ is represented as an n-bit string $(f_0, f_1, \ldots, f_{n-1})$; $g_0 + g_1 t + \cdots + g_{n-1} t^{n-1}$ is represented as an n-bit string $(g_0, g_1, \ldots, g_{n-1})$; and the output $h_0 + h_1 t + \cdots + h_{2n-2} t^{2n-2}$ is represented as a $(2n-1)$-bit string $(h_0, h_1, \ldots, h_{2n-2})$.

Comparison to previous work. The definition of polynomial multiplication immediately implies that $M(n) \le \Theta(n^2)$. Karatsuba showed in [45] that $M(n) \le \Theta(n^{\lg 3})$. Toom showed in [64] that $M(n) \le n^{1+o(1)}$, and more precisely $M(n) \le n2^{\Theta(\sqrt{\lg n})}$. Schönhage showed in [61] that $M(n) \le \Theta(n \lg n \lg \lg n)$, by adapting an integer-multiplication method by Schönhage and Strassen in [62]. No better asymptotic bounds are known; Fürer in [32] introduced an asymptotically faster multiplication method for integers, but it is not clear whether the method can be adapted to binary polynomials.

Of course, bounds involving Θ, O, etc. are not explicit. To draw conclusions about $M(251)$, or any other specific $M(n)$, one needs to carefully re-analyze the algorithms used to prove asymptotic bounds. To draw *useful* conclusions one often needs to rethink the algorithm design, looking for constant-factor (and sub-constant-factor) improvements that are not visible in the asymptotics.

Explicit upper bounds do appear in the literature on hardware multipliers. The bound $M(n) \le 2n^2 - 2n + 1$ is easy to find in hardware textbooks. Several cryptographic-hardware papers (see below) have presented explicit upper bounds on $M(n)$ obtained with Karatsuba's method, have pointed out that these upper bounds are considerably below $2n^2 - 2n + 1$ for cryptographically useful sizes of n, and have concluded that hardware multipliers should use Karatsuba's method. XOR and AND do not have identical hardware costs, but analyses weighted by the actual costs come to similar conclusions.

The explicit upper bounds on $M(n)$ obtained in this section are better than anything that can be found in the hardware literature. Here are several examples of the improved bounds:

- $M(128) \le 11486$. For comparison, Peter and Langendörfer in [57, Table 3] report 14287 bit operations (12100 XORs and 2187 ANDs) for "classic Karatsuba multiplication," 32513 bit operations for schoolbook multiplication, and 13146 bit operations for an "improved iterative Karatsuba."
- $M(163) \le 16923$. For comparison, Chang, Kim, Park, and Lim in [24] report 21791 bit operations for a "non-redundant Karatsuba-Ofman algorithm."
- $M(193) \le 21865$. For comparison, Rodríguez-Henríquez and Koç in [59, Section 4.1] report 29725 bit operations (20524 XORs and 9201 ANDs).
- $M(194) \le 21906$. For comparison, von zur Gathen and Shokrollahi in [67, Section 3] report 26575 bit operations.
- $M(n) \le (103/18)n^{\lg 3} - 7n + 3/2$ for $n \in \{2, 4, 8, 16, \ldots\}$; this bound is not tight. For comparison, Fan, Sun, Gu, and Lam in [29, Table II] report $6.5n^{\lg 3} - 6n + 0.5$ bit operations ($5.5n^{\lg 3} - 6n + 0.5$ XORs and $n^{\lg 3}$ ANDs). Note that $103/18 = 5.722\ldots < 6.5$.
- $M(512) \le 98018$. For comparison, Rodriguez-Henríquez and Koç in [59, Table 1] report 116191 bit operations (81199 XORs and 34992 ANDs).

As these examples illustrate, switching from this paper's multiplication methods back to the multiplication methods in the hardware literature often increases the number of bit operations by more than 20%. On the other hand, one should not exaggerate the importance of multiplication refinements as a component of this paper's new speed records for binary-elliptic-curve cryptography. Bitsliced binary-Edwards-curve arithmetic, as described in Section 3 of this paper, would still have set new speed records even if this section's multiplication methods had been replaced by the methods in [67].

Schoolbook recursion. One can multiply $f_0 + f_1 t + \cdots + f_n t^n$ by $g_0 + g_1 t + \cdots + g_n t^n$ as follows:

- Recursively multiply $f_0 + f_1 t + \cdots + f_{n-1} t^{n-1}$ by $g_0 + g_1 t + \cdots + g_{n-1} t^{n-1}$.
- Compute $(f_n g_0 + f_0 g_n)t^n + (f_n g_1 + f_1 g_n)t^{n+1} + \cdots + f_n g_n t^{2n}$. This takes $2n + 1$ bit multiplications and n bit additions.
- Add. This takes $n - 1$ bit additions for the coefficients of $t^n, \ldots, t^{2n-2}$; the other coefficients do not overlap.

Consequently $M(n+1) \le M(n) + 4n$. This "schoolbook recursion" bound implies the "schoolbook multiplication" bound $M(n) \le 2n^2 - 2n + 1$, but schoolbook recursion is—in combination with other recursions—useful for much larger n's than schoolbook multiplication.

Three-way recursion. The well-known **Karatsuba identity**

$$\begin{aligned}(F_0 + F_1 t^n)(G_0 + G_1 t^n) \\ &= F_0 G_0 + t^n((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1) + t^{2n} F_1 G_1\end{aligned}$$

shows how to multiply $2n$-bit polynomials $F_0 + F_1t^n$, $G_0 + G_1t^n$ using three multiplications of n-bit polynomials and a small amount of overhead. What actually appeared as "Theorem 2 (Karatsuba)" in the original Karatsuba–Ofman paper [45] was an integer analogue of the squaring case of the equivalent identity

$$\begin{aligned}&(F_0 + F_1t^n)(G_0 + G_1t^n)\\&\qquad = (1 - t^n)F_0G_0 + t^n(F_0 + F_1)(G_0 + G_1) + (t^{2n} - t^n)F_1G_1.\end{aligned}$$

Either identity easily leads to the bound $M(2n) \le 3M(n) + 8n - 4$ appearing in, e.g., [67, Section 2]. The $8n - 4$ arises as a sum of three components: $2n$ for the additions in $F_0 + F_1$ and $G_0 + G_1$, another $4n - 2$ to subtract F_0G_0 and F_1G_1 from $(F_0 + F_1)(G_0 + G_1)$, and another $2n - 2$ to handle the overlaps between F_0G_0, $t^n \cdots$, and $t^{2n}F_1G_1$.

Much less well known is that the constant 8 here can be improved to 7: specifically, $M(2n) \le 3M(n) + 7n - 3$. What follows is one way to understand the improvement.

A generic quadratic polynomial $H = H_0 + H_1x + H_2x^2$ can be reconstructed from $H(0) = H_0$, $H(1) = H_0 + H_1 + H_2$, and $H(\infty) = H_2$ by the projective Lagrange interpolation formula $H = (1 - x)H(0) + xH(1) + x(x - 1)H(\infty)$. Factoring out $1 - x$ produces the slightly simpler formula

$$H = (1 - x)(H(0) - xH(\infty)) + xH(1).$$

In particular, if H is the product of two generic linear polynomials $F = F_0 + F_1x$ and $G = G_0 + G_1x$, then $H(0) = F(0)G(0) = F_0G_0$, $H(1) = F(1)G(1) = (F_0+F_1)(G_0+G_1)$, and $H(\infty) = F(\infty)G(\infty) = F_1G_1$, so $(F_0+F_1x)(G_0+G_1x) = (1 - x)(F_0G_0 - xF_1G_1) + x(F_0 + F_1)(G_0 + G_1)$. Substitute $x = t^n$ to obtain the **refined Karatsuba identity**

$$(F_0 + F_1t^n)(G_0 + G_1t^n) = (1 - t^n)(F_0G_0 - t^nF_1G_1) + t^n(F_0 + F_1)(G_0 + G_1).$$

For comparison, rewriting the projective Lagrange interpolation formula as $H = H(0) + x(H(1) - H(0) - H(\infty)) + x^2H(\infty)$ leads to the original Karatsuba identity.

Say F_0, G_0 are n-bit polynomials in $\mathbf{F}_2[t]$ and F_1, G_1 are k-bit polynomials in $\mathbf{F}_2[t]$. Here is a cost analysis of the refined Karatsuba identity as a method of computing the product of $F_0 + F_1t^n$ and $G_0 + G_1t^n$:

- Cost $M(n)$: Multiply F_0 by G_0.
- Cost $M(k)$: Multiply F_1 by G_1.
- Cost k, assuming $k \le n$: Add F_0 to F_1.
- Cost k: Add G_0 to G_1.
- Cost $M(n)$: Multiply $F_0 + F_1$ by $G_0 + G_1$.
- Cost $n - 1$, assuming $k \ge n/2$: Subtract $t^nF_1G_1$ from F_0G_0.
- Cost $2k - 1$: Subtract $t^n(F_0G_0 - t^nF_1G_1)$ from $F_0G_0 - t^nF_1G_1$.
- Cost $2n - 1$: Add $t^n(F_0 + F_1)(G_0 + G_1)$.

Consequently $M(n+k) \le 2M(n) + M(k) + 4k + 3n - 3$ if $n/2 \le k \le n$.

Notice that this computation does not follow the traditional structure of first computing the coefficients H_0, H_1, H_2 and then computing $H(t^n) = H_0 + H_1 t^n + H_2 t^{2n}$. In particular, the middle coefficient $H_1 = (F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1$ is not an intermediate result in this computation.

I posted the $M(2n) \le 3M(n) + 7n - 3$ bound (and the resulting $M(n) \le (103/18)n^{\lg 3} - 7n + 3/2$ bound for $n \in \{2, 4, 8, \ldots\}$) in [11, page 7] in 2000, but I am formally announcing the idea here for the first time. The simplified-Lagrange-interpolation explanation has not appeared anywhere, even informally, and is reused below for improvements in five-way recursion.

Five-way recursion. A generic quartic polynomial $H = H_0 + H_1 x + H_2 x^2 + H_3 x^3 + H_4 x^4$ over $\mathbf{F}_2$ can be reconstructed from the values $H(0)$, $H(1)$, $H(t)$, $H(t+1)$, $H(\infty)$ by the projective Lagrange interpolation formula

$$\begin{aligned} H = H(0)&\frac{(x+1)(x+t)(x+t+1)}{t(t+1)} + H(1)\frac{x(x+t)(x+t+1)}{(1+t)t} \\ &+ H(t)\frac{x(x+1)(x+t+1)}{t(t+1)} + H(t+1)\frac{x(x+1)(x+t)}{(t+1)t} \\ &+ H(\infty)x(x+1)(x+t)(x+t+1). \end{aligned}$$

Easy manual simplification produces the (perhaps not optimal) formula

$$H = U + H(\infty)(x^4 + x) + \frac{(U + V + H(\infty)(t^4 + t))(x^2 + x)}{t^2 + t}$$

where $U = H(0) + (H(0) + H(1))x$ and $V = H(t) + (H(t) + H(t+1))(x+t)$. This formula, in turn, leads to the following new algorithm to compute the product of $F_0 + F_1 t^n + F_2 t^{2n}$ and $G_0 + G_1 t^n + G_2 t^{2n}$, where F_0, F_1, G_0, G_1 are n-bit polynomials and F_2, G_2 are k-bit polynomials:

- Cost $M(n)$: Compute $H(0) = F_0 G_0$.
- Cost $M(k)$: Compute $H(\infty) = F_2 G_2$.
- Cost $n + k$, assuming $k \le n$: Compute $F_0 + F_1 + F_2$.
- Cost $n + k$: Compute $G_0 + G_1 + G_2$.
- Cost $M(n)$: Compute $H(1) = (F_0 + F_1 + F_2)(G_0 + G_1 + G_2)$.
- Cost $n - 1$, or k if $k \le n - 1$: Compute $F_1 t + F_2 t^2$.
- Cost $n - 1$, or k if $k \le n - 1$: Compute $G_1 t + G_2 t^2$.
- Cost $n - 1$: Compute $F_0 + (F_1 t + F_2 t^2)$.
- Cost $n - 1$: Compute $G_0 + (G_1 t + G_2 t^2)$.
- Cost $M(n+2)$, or $M(n+1)$ if $k \le n - 1$: Compute $H(t)$, the product of $F_0 + (F_1 t + F_2 t^2)$ and $G_0 + (G_1 t + G_2 t^2)$.
- Cost $n - 1$: Compute $(F_0 + F_1 + F_2) + (F_1 t + F_2 t^2)$.
- Cost $n - 1$: Compute $(G_0 + G_1 + G_2) + (G_1 t + G_2 t^2)$.
- Cost $M(n+2)$, or $M(n+1)$ if $k \le n - 1$: Compute $H(t+1)$, the product of $(F_0 + F_1 + F_2) + (F_1 t + F_2 t^2)$ and $(G_0 + G_1 + G_2) + (G_1 t + G_2 t^2)$.

- Cost $2n+1$: Compute $H(t)+H(t+1)$. The coefficients of t^{2n+2} and t^{2n+1} in $H(t+1)$ are the same as the coefficients of t^{2n+2} and t^{2n+1} in $H(t)$, so this sum has degree at most $2n$. (For the same reason, some work could have been saved in the computation of $H(t+1)$.)
- Cost $3n+4$, or $3n+2$ if $k \le n-1$: Compute $V = H(t)+(H(t)+H(t+1))(t^n+t)$. Note that $\deg V \le 3n$.
- Cost $3n-2$: Compute $U = H(0)+(H(0)+H(1))t^n$. Note that $\deg U \le 3n-2$.
- Cost $4k+3n-3$, assuming $n \ge 2$: Compute $W = U+V+H(\infty)(t^4+t)$. Note that $\deg W \le 3n$.
- Cost $3n-2$: Compute $W/(t^2+t)$. This division is exact: $W/(t^2+t) \in \mathbf{F}_2[t]$. (For the same reason, some work could have been skipped in the computation of W.)
- Cost $5n+2k-4$: Compute $H(\infty)(t^{4n}+t^n)+(W/(t^2+t))(t^{2n}+t^n)+U$.

Consequently $M(3n) \le 3M(n)+2M(n+2)+35n-12$ if $n \ge 2$, and $M(2n+k) \le 2M(n)+M(k)+2M(n+1)+25n+10k-12$ if $1 \le k \le n-1$.

The previous state of the art, building on ideas by Zimmermann and Quercia, Weimerskirch and Paar [68], and Montgomery [53], was an algorithm by Bodrato in [19] to compute H_0, H_1, H_2, H_3, H_4 given $H(0), H(1), H(t), H(t+1), H(\infty)$ using 9 additions, one multiplication by t^3+1, one division by t, one division by $t+1$, and one division by t^2+t. The total overhead was about $38n$ operations: $10n$ to compute $F(0), F(1), F(t), F(t+1), F(\infty)$ and $G(0), G(1), G(t), G(t+1), G(\infty)$; $24n$ for Bodrato's computation of H_0, H_1, H_2, H_3, H_4; and $4n$ to reconstruct H from H_0, H_1, H_2, H_3, H_4. The separate coefficients H_0, H_1, H_2, H_3, H_4 turn out to be a distraction, as in the Karatsuba case; this section does better by constructing H directly, exploiting the polynomial structure visible in the projective Lagrange interpolation formula.

Two-level seven-way recursion. Consider the problem of multiplying two degree-3 polynomials. Apply the refined Karatsuba identity three times, factor out $1-x$, and substitute $x = t^n$, to obtain the identity

$$\begin{aligned}(F_0 + F_1t^n + F_2t^{2n} + F_3t^{3n})&(G_0 + G_1t^n + G_2t^{2n} + G_3t^{3n})\\ = (1-t^{2n})((1-t^n)&(F_0G_0 - t^nF_1G_1 - t^{2n}F_2G_2 + t^{3n}F_3G_3)\\ &+ t^n(F_0+F_1)(G_0+G_1) - t^{3n}(F_2+F_3)(G_2+G_3))\\ &+ t^{2n}(F_0+F_2+(F_1+F_3)t^n)(G_0+G_2+(G_1+G_3)t^n).\end{aligned}$$

Cost evaluation for polynomials with $3n+k$ coefficients, assuming $k \ge n/2$:

- Cost $M(n)$: Multiply F_0 by G_0.
- Cost $M(n)$: Multiply F_1 by G_1.
- Cost $M(n)$: Multiply F_2 by G_2.
- Cost $M(k)$: Multiply F_3 by G_3.
- Cost $3n-3$: Compute $U = F_0G_0 - t^nF_1G_1 - t^{2n}F_2G_2 + t^{3n}F_3G_3$.
- Cost $2n+2k-1$: Compute $(1-t^n)U$.
- Cost $2n+M(n)$: Multiply F_0+F_1 by G_0+G_1.
- Cost $2k+M(n)$: Multiply F_2+F_3 by G_2+G_3.

- Cost $4n-2$: Compute $V=(1-t^n)U+t^n(F_0+F_1)(G_0+G_1)-t^{3n}(F_2+F_3)(G_2+G_3)$.
- Cost $2n+2k+M(2n)$: Multiply $F_0+F_2+(F_1+F_3)t^n$ by $G_0+G_2+(G_1+G_3)t^n$.
- Cost $6n+2k-2$: Compute $(1-t^{2n})V+t^{2n}(F_0+F_2+(F_1+F_3)t^n)(G_0+G_2+(G_1+G_3)t^n)$.

Hence $M(3n+k)\le M(2n)+5M(n)+M(k)+19n+8k-8$ if $n/2\le k\le n$. For example, $M(4n)\le M(2n)+6M(n)+27n-8$. This is $n-1$ smaller than what would have been obtained by straightforwardly applying the refined Karatsuba identity without factoring out $1-x$.

Optimization. One can build a table of upper bounds on $M(1),\ldots,M(n)$ by recursively building a table of upper bounds on $M(1),\ldots,M(n-1)$ and then mechanically checking what the inequalities in this section say about $M(n)$. This computation reaches $M(251)$ in negligible time. One can slightly improve many of the upper bounds by mechanically removing redundant computations (such as the equal top coefficients of $H(t)$ and $H(t+1)$ in five-way recursion) from straight-line multiplication code.

My web page http://binary.cr.yp.to/m.html presents a table of upper bounds on $M(1),\ldots,M(1000)$ obtained in this way. Each upper bound is accompanied by straight-line multiplication code that has been computer-verified to multiply correctly and to use exactly the specified number of bit operations.

Often an input to one multiplication is reused in a subsequent multiplication; for example, w_1 in Section 3 participates in many multiplications. One can save time by caching evaluations of that input, such as F_0+F_1 above. To properly optimize this reuse one should define, e.g., $M_2(n)$ as the cost of multiplying a single n-bit input by two n-bit inputs (serially), and then optimize $M_2(n)$ analogously to $M(n)$.

The reader is cautioned that there are many more multiplication methods in the literature: for example, more Toom variants, FFTs, etc. Analyzing, refining, and combining these methods would improve the bounds on $M(n)$ for many integers n, perhaps including $n=251$. Most of the relevant methods are surveyed in [22] but have not yet been optimized for bit operations.

3 Elliptic-curve scalar multiplication

This section reviews binary Edwards curves; discusses this paper's selection of a particular binary Edwards curve; and analyzes the speed of computation of scalar multiples on that curve.

Review of binary Edwards curves. A **binary Edwards curve** over a binary finite field k is a curve of the form $d_1(x+y)+d_2(x^2+y^2)=(x+x^2)(y+y^2)$ where $d_1\in k-\{0\}$ and $d_2\in k-\left\{d_1^2+d_1\right\}$. This curve shape was introduced by Bernstein, Lange, and Rezaeian Farashahi in [16] as a characteristic-2 analogue to the curve shape introduced by Edwards in [28].

A binary Edwards curve is **complete** if d_2 has trace 1, i.e., if d_2 cannot be written as c^2+c for any $c\in k$. The paper [16] proves that every ordinary elliptic curve over k is birationally equivalent over k to a complete binary Edwards curve if $\#k \geq 8$, and that various addition formulas on the complete binary Edwards curve have no exceptional cases.

The case $d_1 = d_2$ allows various speedups presented in [16]. For example, it allows differential addition and doubling—a single step in a "Montgomery ladder"—using four squarings in k, two multiplications by d_1, and five more multiplications in k. The general case would need four squarings in k, four multiplications by parameters, and six more multiplications in k.

The selected curve. Define $k = \mathbf{F}_{2^{251}} = \mathbf{F}_2[t]/(t^{251} + t^7 + t^4 + t^2 + 1)$. Define $d \in k$ as $t^{57} + t^{54} + t^{44} + 1$; note that d has trace 1. Define E as the binary Edwards curve $d(x + x^2 + y + y^2) = (x + x^2)(y + y^2)$.

This curve is birationally equivalent to the Weierstrass curve $v^2 + uv = u^3 + (d^2 + d)u^2 + d^8$ by $(x, y) \mapsto (u, v) = (d^3(x + y)/(xy + d(x + y)), d^3(x/(xy + d(x + y)) + d + 1))$. See [16, Section 2].

The main task considered here is scalar multiplication $n, P \mapsto nP$ in the group $E(k) = \left\{(x, y) \in k^2 : d(x + x^2 + y + y^2) = (x + x^2)(y + y^2)\right\}$, with neutral element $(0, 0)$. Note that this group does not have any points at infinity. See [16] for further discussion of the group law.

Security issues in curve selection. This curve satisfies all of the standard criteria for high-security curves:

- The curve has near-prime order. Specifically, the curve has order $4p_1$ where p_1 is the prime $2^{249} + 17672450755679567125975931502191870417$.
- The twist of the curve has near-prime order, specifically order $2p_2$ where p_2 is the prime $2^{250} - 35344901511359134251951863004383740833$.
- The primes are large enough for high security: generic discrete-logarithm algorithms use approximately 2^{124} group operations on average.
- Avoiding subfields: The j-invariant $1/d^8$ generates the field k.
- Avoiding small discriminants: $(2^{251} + 1 - 4p_1)^2 - 2^{253}$ is divisible by the large prime $((2^{251} + 1 - 4p_1)^2 - 2^{253})/(-83531196553759)$ exactly once.
- Avoiding pairing attacks: The multiplicative order of 2^{251} modulo p_1 is not small: in fact, it is $(p_1 - 1)/2$. The multiplicative order of 2^{251} modulo p_2 is not small: in fact, it is $(p_2 - 1)/2$.
- Avoiding the GHS attack: The extension degree 251 is a prime, so the only nontrivial subfield of k is $\mathbf{F}_2$. GHS genera over $\mathbf{F}_2$ cannot be small: in fact, they are at least 2^{49}, since the multiplicative order of 2 modulo 251 is 50. See [51].

The curve used in this paper was found by a search through various possibilities for d. Most choices of d fail the near-prime-order requirement, and most of the remaining choices of d fail the near-prime-twist-order requirement, but there are still many suitable possibilities. An easy computation with the Magma computer-algebra system [20] located a few suitable trinomials, many suitable quadrinomials, etc. The first trinomial found was $t^{141} + t^{28} + 1$, the first quadrinomial found was $t^{57} + t^{54} + t^{44} + 1$ (although $(t^{222} + 1)(t^{21} + 1)$ is an interesting alternative), and the first pentanomial found was $t^{23} + t^{16} + t^{15} + t + 1$.

Some standards omit, or weaken, some of the criteria listed above, for several reasons:

- Some of the criteria do not have known benefits. For example, subfield curves produce only a small loss of security, which can be corrected by a small increase in field size. The small-discriminant criterion is even more difficult to defend; there is no known attack that exploits small discriminants, and there are reasons to guess that randomly chosen large-discriminant curves are *more dangerous* than small-discriminant curves. See [47, Sections 11.1–11.3].
- Sometimes the criteria have disadvantages. For example, some criteria have to be weakened by anyone who wants to allow curves with special algebraic structures, such as "pairing-friendly curves," "Koblitz curves," "Gallant–Lambert–Vanstone curves," and the new "Galbraith–Lin–Scott curves." See, e.g., [34], [33], and [39].
- One of the criteria, twist security, has protocol-level benefits that were not visible in the traditional study of the elliptic-curve discrete-logarithm problem. Twist security has, as a result, often been neglected even in situations where it has no disadvantages. For further discussion of twist security see [44, Section 4], [21, Section 4], [25, Section 4.1], [12, Section 3], and [31, Section 5].

This paper's selection of a curve meeting all the security criteria should not be interpreted as criticism of curves that meet fewer security criteria. One should expect some of those curves, when combined with the techniques in this paper, to achieve even better speeds than the speeds reported in this paper.

Speed issues in curve selection. Even within the restricted pool of curves meeting all of the security criteria discussed above, there are still considerable variations in speed. Standard practice is to focus on the highest-speed curves.

In particular, the fastest elliptic-curve-scalar-multiplication methods involve many multiplications by curve coefficients; it is standard practice to choose these coefficients to be "small." The exact definition of "small" varies but is aimed at speeding up multiplications by these coefficients. For example:

- Weierstrass curves: IEEE Standard P1363 chooses curves $y^2 = x^3 - 3x + b$ to "provide the fastest arithmetic on elliptic curves"; see [2, Section A.9]. Chudnovsky and Chudnovsky had pointed out in [26] that choosing a small coefficient a in $y^2 = x^3 + ax + b$ saves time in elliptic-curve scalar multiplication, and that the particular choice $a = -3$ saves even more time. NIST's standard curves were chosen by the recipe specified in IEEE Standard P1363; see [1, Appendix 6, Section 1.4].
- Montgomery curves: The curve "Curve25519" specified in [12] is the curve $y^2 = x^3 + 486662x^2 + x$ modulo $2^{255} - 19$. Montgomery had pointed out in [52] that his fast differential-addition formulas for $y^2 = x^3 + ax^2 + x$ involve multiplications by $(a + 2)/4$ and benefit from $(a + 2)/4$ being small.
- Binary Edwards curves: [16] makes the analogous suggestion to choose a small parameter d for the curve $d(x + x^2 + y + y^2) = (x + x^2)(y + y^2)$.

This paper chooses $d = t^{57} + t^{54} + t^{44} + 1$, combining a small degree with a small number of terms. Multiplication of a 251-bit polynomial by the quadrinomial $t^{57} + t^{54} + t^{44} + 1$ in $\mathbf{F}_2[t]$ uses only $3 \cdot 251 - 57 = 696$ bit operations, and reduction of the 308-bit product modulo $t^{251} + t^7 + t^4 + t^2 + 1$ uses only 200 bit operations (slightly better than the obvious bound $4 \cdot 57 = 228$ since $t^4, t^2, 1$ are evenly spaced), for a total of just 896 bit operations to multiply by d.

Differential addition and doubling on binary Edwards curves. The following formulas are the "affine $d_1 = d_2$" and "mixed $d_1 = d_2$" formulas from [16, Section 7], repeated here to keep this paper self-contained.

For each point $P = (x, y) \in E(k)$ define $w(P) = x + y$. Then $w(2P) = 1 + d/(d + w(P)^2 + w(P)^4)$, and more generally $w(Q + P) + w(Q - P) = 1 + d/(d + w(P)w(Q)(1 + w(P))(1 + w(Q)))$. The denominators here are never zero.

The following formulas use four squarings, two multiplications by d, and five more multiplications to compute $w(2P), w(Q + P)$ as fractions $W_4/Z_4, W_5/Z_5$, given $w(P), w(Q)$ as fractions $W_2/Z_2, W_3/Z_3$ and given $w(Q - P)$ as an element $w_1 \in k$:

$$\begin{aligned} C &= W_2 \cdot (Z_2 + W_2); & W_4 &= C^2; & Z_4 &= d(Z_2^2)^2 + W_4; \\ V &= C \cdot W_3 \cdot (Z_3 + W_3); & Z_5 &= V + d(Z_2 \cdot Z_3)^2; & W_5 &= V + Z_5 \cdot w_1. \end{aligned}$$

This operation is called **mixed differential addition and doubling**.

Conditional swaps. This paper uses a scalar-multiplication strategy introduced by Montgomery in [52, Section 10.3.1], often called the "Montgomery ladder." The most important step is **conditionally swapped mixed differential addition and doubling**. This means computation of $w(2P), w(Q + P)$ as fractions $W_4/Z_4, W_5/Z_5$ if $\beta = 0$, and computation of $w(P + Q), w(2Q)$ as fractions $W_4/Z_4, W_5/Z_5$ if $\beta = 1$. The inputs are $w(P), w(Q)$ as fractions $W_2/Z_2, W_3/Z_3$ as above; $w(Q - P)$ as an element $w_1 \in k$ as above; and an extra bit $\beta \in \{0, 1\}$.

The standard way to handle $\beta = 1$ is to first swap $W_2/Z_2, W_3/Z_3$, then proceed with the original computation, and finally swap $W_4/Z_4, W_5/Z_5$. The first swap exactly reverses the roles of P and Q, since $w(P - Q) = w(Q - P)$; the original computation therefore produces $w(2Q), w(P + Q)$; and the final swap produces $w(P + Q), w(2Q)$ as desired.

The standard way to swap W_2 and W_3 conditionally on β without branching is to replace (W_2, W_3) by $(W_2 + \beta(W_3 - W_2), W_3 - \beta(W_3 - W_2))$. Similar comments apply to (Z_2, Z_3), (W_4, W_5), and (Z_4, Z_5).

Scalar multiplication. The last step in scalar multiplication computes $w(nP)$, $w(nP + P)$ as fractions, starting from $w(\lfloor n/2 \rfloor P), w(\lfloor n/2 \rfloor P + P)$ as fractions and $w(P)$ as an element of k. This is an example of conditionally swapped mixed differential addition and doubling, where β is the bottom bit of n.

The previous step produces $w(\lfloor n/2 \rfloor P), w(\lfloor n/2 \rfloor P + P)$ as fractions starting from $w(\lfloor n/4 \rfloor P), w(\lfloor n/4 \rfloor P + P)$ as fractions and $w(P)$ as an element of k. This is the same computation, except that n is replaced by $\lfloor n/2 \rfloor$; i.e., β is the second bit of n. The conditional swap at the end of this step is followed immediately by, and can be profitably merged with, the conditional swap at the beginning of the next step.

Similar comments apply to earlier steps. If the target scalar n is known to be between 0 and $2^b - 1$ then one can use a sequence of b steps. The first step produces $w(\lfloor n/2^{b-1} \rfloor P), w(\lfloor n/2^{b-1} \rfloor P + P)$ from $w(\lfloor n/2^{b} \rfloor P), w(\lfloor n/2^{b} \rfloor P + P)$; i.e., from $0, w(P)$.

To summarize: Given $w(P) \in k$ and a b-bit scalar n, this scalar-multiplication method uses b conditionally swapped mixed differential additions and doublings to produce (W, Z) such that $W/Z = w(nP)$. The $2b$ conditional swaps can be merged into just $b + 1$ conditional swaps.

Postprocessing. One can divide W by Z, obtaining $w(nP) \in k$, by computing $WZ^{2^{251}-2}$ with (e.g.) 250 squarings and 11 more multiplications. The multiplications produce Z^3, Z^7, Z^{2^6-1}, $Z^{2^{12}-1}$, $Z^{2^{24}-1}$, $Z^{2^{25}-1}$, $Z^{2^{50}-1}$, $Z^{2^{100}-1}$, $Z^{2^{125}-1}$, $Z^{2^{250}-1}$, and $WZ^{2^{251}-2}$.

A small extra computation shown in [16, Section 7], using the x and y coordinates of P separately, would produce the x and y coordinates of nP separately; but the w coordinate is adequate for elliptic-curve Diffie–Hellman. One can also check directly that w corresponds to a curve point by checking that $d/(w + w^2)$ has trace 0 and that $w + w^2$ times the half-trace of $d/(w + w^2)$ has trace 0. These computations are much faster than scalar multiplication and are not discussed further in this paper.

Instead of inverting Z at the end of the computation one can use affine coordinates, eliminating some multiplications at the cost of an inversion in each step. Inversions are well known to benefit from batching, thanks to "Montgomery's trick" from [52, page 260]. However, each inversion still costs slightly more than 3 multiplications, wiping out most if not all of the gain. Montgomery in [52, page 261] compared batched affine coordinates to projective coordinates in the non-binary case and reported negligible performance differences. One should not expect affine coordinates to provide large savings in the binary case.

Performance: bit operations. A 251-bit single-scalar multiplication as described here involves 44679665 bit operations; this number has been computer-verified. The main cost is 43011084 bit operations for 1266 field multiplications (1255 in the main loop and 11 in the final division). Each multiplication uses 33974 bit operations: 33096 bit operations for 251-bit multiplication in $\mathbf{F}_2[t]$, and 878 bit operations to reduce the 501-bit product modulo $t^{251} + t^7 + t^4 + t^2 + 1$. The other costs are as follows:

- 397518 bit operations for 1254 squarings (1004 in the main loop and 250 in the final division), each using 317 bit operations;
- 449792 bit operations for 502 multiplications by d, each using 896 bit operations;
- 315005 bit operations for 1255 additions, each using 251 bit operations; and
- 506266 bit operations for 1004 conditional swaps, which at the cost of 250 bit operations are merged into 504 conditional swaps, each costing 1004 bit operations.

In some protocols the 251-bit scalar is always a multiple of 4, allowing slight speedups. In other protocols a 249-bit scalar is adequate, allowing slight further speedups.

Performance: cycles. The `BBE251` software reads a batch of scalars $n_1, \ldots, n_{128}$ and a batch of curve points $P_1, \ldots, P_{128}$ and computes a batch of multiples $n_1 P_1, \ldots, n_{128} P_{128}$. Each scalar is represented as a 32-byte string in little-endian form. Each curve point is represented as a field element w as described in the previous section; w is, in turn, represented as a 32-byte string.

One might think that writing fast software for this computation on a Core 2 CPU is a simple matter of generating code for the bit operations described in this paper, replacing XORs and ANDs with a C compiler's intrinsic `_mm_xor_si128` and `_mm_and_si128` operations on 128-bit vectors. A single core of the CPU can carry out three of the corresponding `PXOR` and `PAND` instructions per cycle, so 44 million bit operations should be completed in about 15 million cycles—under 120000 cycles per input. Transposing the n's and P's into bitsliced form, and transposing the results out of bitsliced form, takes negligible time.

There is, however, a critical bottleneck in any straightforward implementation: namely, load throughput. In one cycle the CPU can carry out three operations on six vectors *in registers*; but loading those six vectors from memory into registers costs six cycles—the Core 2 performs only one load in each cycle. The results of the three operations are ready to be used for further operations in the next cycle, so one can imagine loading (e.g.) 56 input vectors for a 28-bit multiplication, carrying out all 956 bit operations for the multiplication, and then storing the final outputs; but the Core 2 has only 16 128-bit vector registers.

Recursive multiplication methods such as Karatsuba's method might seem to be ideal for reducing loads and stores, since they split larger multiplication problems into smaller multiplication problems that fit into registers. However, the first step in decomposing a $2n$-bit multiplication into n-bit multiplications is to add $2n$ vectors to another $2n$ vectors—and the $2n/3$ cycles for these additions are swamped by $4n$ cycles for loads. Similar comments apply to the recombination of $(2n-1)$-bit products into a $(4n-1)$-bit product.

Further contributing to the memory pressure is the fact that the Core 2's vector instructions are two-operand instructions such as "replace a with $a+b$," not three-operand instructions such as "replace c with $a+b$." Copying a to c, in situations where a and b need to be reused, takes away one of the three `XOR`/`AND` slots available in a cycle. Copying a to c via memory uses an extra load.

`BBE251` takes several measures to reduce the number of loads and stores. Most importantly, it merges decompositions and recombinations across multiple layers of recursion, reusing sums while they are still in registers. As a simple example, adding $(a_0, \ldots, a_{2n-1})$ to $(a_{2n}, \ldots, a_{4n-1})$ takes $4n$ loads and $2n$ additions; subsequently adding $(a_0, \ldots, a_{n-1})$ to $(a_n, \ldots, a_{2n-1})$, adding $(a_{2n}, \ldots, a_{3n-1})$ to $(a_{3n}, \ldots, a_{4n-1})$, and adding $(a_0 + a_{2n}, \ldots, a_{n-1} + a_{3n-1})$ to $(a_n + a_{3n}, \ldots, a_{2n-1} + a_{4n-1})$ would take $6n$ loads and $3n$ additions; but performing all of these operations together reduces the $10n$ loads to $4n$ loads and $2n$ copies.

The current version of `BBE251` merges most operations across two levels of recursion, and takes fewer than 44 million cycles, although still many more than the target of 15 million. It is not yet clear how close the correlations are between optimized bit-operation counts and optimized cycle counts, but it is clear that schoolbook multiplication could not have been competitive with `BBE251`. Larger-scale load/store elimination is underway and can be expected to further improve `BBE251`'s performance.

References

1. — (no editor), *Digital signature standard (DSS)*, Federal Information Processing Standard 186-2, National Institute of Standards and Technology, 2000. URL: http://csrc.nist.gov/publications/fips/. Citations in this document: §3.
2. — (no editor), *Standard specifications for public key cryptography*, IEEE, 2000. Citations in this document: §3.
3. — (no editor), *Information theory workshop, 2006. ITW '06 Chengdu*, IEEE, 2006. See [67].
4. — (no editor), *SPEED: software performance enhancement for encryption and decryption*, 2007. URL: http://www.hyperelliptic.org/SPEED. See [35].
5. — (no editor), *Design, automation & test in Europe conference & exhibition, 2007. DATE '07*, IEEE, 2007. See [57].
6. — (no editor), *Fifth international conference on information technology: new generations (ITNG 2008), 7–8 April 2008, Las Vegas, Nevada, USA*, IEEE, 2008. See [37].
7. — (no editor), *Fifth workshop on fault diagnosis and tolerance in cryptography (FDTC '08)*, IEEE, 2008. See [31].
8. Kazumaro Aoki, Fumitaka Hoshino, Tetsutaro Kobayashi, *A cyclic window algorithm for ECC defined over extension fields*, in [58] (2001), 62–73. Citations in this document: §1.
9. Kazumaro Aoki, Fumitaka Hoshino, Tetsutaro Kobayashi, Hiroaki Oguro, *Elliptic curve arithmetic using SIMD*, in [27] (2001), 235–247. Citations in this document: §1, §1.
10. Daniel V. Bailey, Christof Paar, *Efficient arithmetic in finite field extensions with application in elliptic curve cryptography*, Journal of Cryptology **14** (2001), 153–176. ISSN 0933–2790. Citations in this document: §1.
11. Daniel J. Bernstein, *Fast multiplication* (2000). URL: http://cr.yp.to/talks.html#2000.08.14. Citations in this document: §2.
12. Daniel J. Bernstein, *Curve25519: new Diffie-Hellman speed records*, in [69] (2006), 207–228. URL: http://cr.yp.to/papers.html#curve25519. Citations in this document: §1, §1, §1, §3, §3.
13. Daniel J. Bernstein, *Can we avoid tests for zero in fast elliptic-curve arithmetic?* (2006). URL: http://cr.yp.to/papers.html#curvezero. Citations in this document: §1.
14. Daniel J. Bernstein, Tanja Lange, *Faster addition and doubling on elliptic curves*, in [49] (2007), 29–50. URL: http://cr.yp.to/papers.html#newelliptic. Citations in this document: §1.
15. Daniel J. Bernstein, Tanja Lange (editors), *eBACS: ECRYPT Benchmarking of Cryptographic Systems*, accessed 3 June 2009 (2009). URL: http://bench.cr.yp.to. Citations in this document: §1.
16. Daniel J. Bernstein, Tanja Lange, Reza Rezaeian Farashahi, *Binary Edwards curves*, in [55] (2008), 244–265. URL: http://cr.yp.to/papers.html#edwards2. Citations in this document: §1, §1, §1, §3, §3, §3, §3, §3, §3, §3, §3.
17. Eli Biham (editor), *Fast Software Encryption '97*, Lecture Notes in Computer Science, 1267, Springer-Verlag, 1997. ISBN 3-540-63247-6. See [18].
18. Eli Biham, *A fast new DES implementation in software*, in [17] (1997), 260–272. Citations in this document: §1.
19. Marco Bodrato, *Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0*, in [23] (2007), 116–133. URL: http://bodrato.it/papers/#WAIFI2007. Citations in this document: §2.
20. Wieb Bosma, John Cannon, Catherine Playoust, *The Magma algebra system. I. The user language*, Journal of Symbolic Computation **24** (1997), 235–265. Citations in this document: §3.
21. Colin Boyd, Paul Montague, Khanh Nguyen, *Elliptic curve based password authenticated key exchange protocols*, in [66] (2001), 487–501. URL: http://sky.fit.qut.edu.au/~boydc/papers/. Citations in this document: §3.
22. Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, Paul Zimmermann, *Faster multiplication in GF(2)[x]*, in [65], 153–166. URL: http://wwwmaths.anu.edu.au/~brent/pub/pub232.html. Citations in this document: §1, §2.
23. Claude Carlet, Berk Sunar (editors), *Arithmetic of finite fields, first international workshop, WAIFI 2007, Madrid, Spain, June 21–22, 2007, proceedings*, Lecture Notes in Computer Science, 4547, Springer, 2007. ISBN 978-3-540-73073-6. See [19].
24. Nam Su Chang, Chang Han Kim, Young-Ho Park, Jongin Lim, *A non-redundant and efficient architecture for Karatsuba-Ofman algorithm*, in [70] (2005), 288–299. Citations in this document: §2.
25. Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, David Pointcheval, *The Twist-AUgmented technique for key exchange*, in [69] (2006), 410–426. URL: http://www.loria.fr/~gaudry/papers.en.html. Citations in this document: §3.
26. David V. Chudnovsky, Gregory V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, Advances in Applied Mathematics **7** (1986), 385–434. MR 88h:11094. Citations in this document: §3.
27. George I. Davida, Yair Frankel (editors), *Information security, 4th international conference, ISC 2001, Malaga, Spain, October 1–3, 2001, proceedings*, Lecture Notes in Computer Science, 2200, Springer, 2001. ISBN 978-3-540-42662-2. See [9].
28. Harold M. Edwards, *A normal form for elliptic curves*, Bulletin of the American Mathematical Society **44** (2007), 393–422. URL: http://www.ams.org/bull/2007-44-03/S0273-0979-07-01153-6/home.html. Citations in this document: §3.
29. Haining Fan, Jiaguang Sun, Ming Gu, Kwok-Yan Lam, *Overlap-free Karatsuba-Ofman polynomial multiplication algorithms for hardware implementations*, 7 Oct 2008 version (2008). URL: http://eprint.iacr.org/2007/393. Citations in this document: §2.
30. Kenny Fong, Darrel Hankerson, Julio López, Alfred Menezes, *Field inversion and point halving revisited*, IEEE Transactions on Computers **53** (2004), 1047–1059. ISSN 0018–9340. URL: http://www.cacr.math.uwaterloo.ca/techreports/2003/tech_reports2003.html. Citations in this document: §1.
31. Pierre-Alain Fouque, Reynald Lercier, Denis Réal, Frédéric Valette, *Fault attack on elliptic curve with Montgomery ladder implementation*, in [7] (2008), 92–98. URL: http://www.di.ens.fr/~fouque/index-pub.html. Citations in this document: §3.
32. Martin Fürer, *Faster integer multiplication*, in [42] (2007), 57–66. URL: http://www.cse.psu.edu/~furer/. Citations in this document: §2.
33. Steven Galbraith, Xibin Lin, Michael Scott, *Endomorphisms for faster elliptic curve cryptography on a large class of curves*, in [43] (2009), 518–535. URL: http://eprint.iacr.org/2008/194. Citations in this document: §1, §3.
34. Robert P. Gallant, Robert J. Lambert, Scott A. Vanstone, *Faster point multiplication on elliptic curves with efficient endomorphisms*, in [46] (2001), 190–200. MR 2003h:14043. Citations in this document: §3.

35. Pierrick Gaudry, Emmanuel Thomé, *The mpFq library and implementing curve-based key exchanges*, in [4] (2007), 49–64. URL: http://www.loria.fr/~gaudry/papers.en.html. Citations in this document: §1, §1, §1.
36. Tim Güneysu, Christof Paar, *Ultra high performance ECC over NIST primes on commercial FPGAs*, in [55] (2008), 62–78. Citations in this document: §1, §1.
37. Shay Gueron, Michael E. Kounavis, *A technique for accelerating characteristic 2 elliptic curve cryptography*, in [6] (2008), 265–272. Citations in this document: §1.
38. Darrel Hankerson, Julio Lopez Hernandez, Alfred Menezes, *Software implementation of elliptic curve cryptography over binary fields*, in [48] (2000), 1–24. URL: http://www.cacr.math.uwaterloo.ca/techreports/2000/corr2000-42.ps. Citations in this document: §1, §1.
39. Darrel Hankerson, Koray Karabina, Alfred Menezes, *Analyzing the Galbraith–Lin–Scott point multiplication method for elliptic curves over binary fields* (2008). URL: http://eprint.iacr.org/2008/334. Citations in this document: §1, §3.
40. Intel Corporation, *Carry-less multiplication and its usage for computing the GCM mode* (2008). URL: http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode. Citations in this document: §1.
41. Intel Corporation, *Intel Advanced Vector Extensions programming reference*, 2008. URL: http://softwarecommunity.intel.com/isn/downloads/intelavx/Intel-AVX-Programming-Reference-31943302.pdf. Citations in this document: §1.
42. David S. Johnson, Uriel Feige (editors), *Proceedings of the 39th annual ACM symposium on theory of computing, San Diego, California, USA, June 11–13, 2007*, Association for Computing Machinery, New York, 2007. ISBN 978–1–59593–631–8. See [32].
43. Antoine Joux (editor), *Advances in Cryptology—EUROCRYPT 2009, 28th annual international conference on the theory and applications of cryptographic techniques, Cologne, Germany, April 26–30, 2009, proceedings*, Lecture Notes in Computer Science, 5479, Springer, 2009. ISBN 978-3-642-01000-2. See [33].
44. Burton S. Kaliski Jr., *One-way permutations on elliptic curves*, Journal of Cryptology **3** (1991), 187–199. Citations in this document: §3.
45. Anatoly A. Karatsuba, Y. Ofman, *Multiplication of multidigit numbers on automata*, Soviet Physics Doklady **7** (1963), 595–596. ISSN 0038–5689. URL: http://cr.yp.to/bib/entries.html#1963/karatsuba. Citations in this document: §2, §2.
46. Joe Kilian (editor), *Advances in cryptology: CRYPTO 2001, 21st annual international cryptology conference, Santa Barbara, California, USA, August 19–23, 2001, proceedings*, Lecture Notes in Computer Science, 2139, Springer, 2001. ISBN 3-540-42456-3. MR 2003d:94002. See [34].
47. Ann Hibner Koblitz, Neal Koblitz, Alfred Menezes, *Elliptic curve cryptography: the serpentine course of a paradigm shift* (2008). URL: http://eprint.iacr.org/2008/390. Citations in this document: §3.
48. Çetin Kaya Koç, Christof Paar (editors), *Cryptographic hardware and embedded systems—CHES 2000: Proceedings of the 2nd International Workshop held in Worcester, MA, USA, August 2000*, Lecture Notes in Computer Science, 1965, Springer, 2000. ISBN 3-540-42521-7. See [38].
49. Kaoru Kurosawa (editor), *Advances in cryptology—ASIACRYPT 2007, 13th international conference on the theory and application of cryptology and information security, Kuching, Malaysia, December 2–6, 2007, proceedings*, Lecture Notes in Computer Science, 4833, Springer, 2007. ISBN 978-3-540-76899-9. See [14].
50. Mitsuru Matsui, Junko Nakajima, *On the power of bitslice implementation on Intel Core2 processor*, in [56] (2007), 121–134. Citations in this document: §1.
51. Alfred Menezes, Minghua Qu, *Analysis of the Weil descent attack of Gaudry, Hess and Smart*, in [54] (2001), 308–318. Citations in this document: §3.
52. Peter L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), 243–264. ISSN 0025–5718. MR 88e:11130. URL: http://links.jstor.org/sici?sici=0025-5718(198701)48:177<243:STPAEC>2.0.CO;2-3. Citations in this document: §1, §3, §3, §3, §3.
53. Peter L. Montgomery, *Five, six, and seven-term Karatsuba-like formulae*, IEEE Transactions on Computers **54** (2005), 362–369. Citations in this document: §2.
54. David Naccache (editor), *Topics in cryptology—CT-RSA 2001: Proceedings of the Cryptographers' Track at the RSA Conference held in San Francisco, CA, April 8–12, 2001*, Lecture Notes in Computer Science, 2020, Springer, 2001. ISBN 3-540-41898-9. MR 2003a:94039. See [51].
55. Elisabeth Oswald, Pankaj Rohatgi (editors), *Cryptographic hardware and embedded systems—CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10–13, 2008, Proceedings*, Lecture Notes in Computer Science, 5154, Springer, 2008. ISBN 978-3-540-85052-6. See [16], [36].
56. Pascal Paillier, Ingrid Verbauwhede (editors), *Cryptographic hardware and embedded systems—CHES 2007, 9th international workshop, Vienna, Austria, September 10–13, 2007, proceedings*, Lecture Notes in Computer Science, 4727, Springer, 2007. ISBN 978-3-540-74734-5. See [50].
57. Steffen Peter, Peter Langendörfer, *An efficient polynomial multiplier in GF(2^m) and its application to ECC designs*, in [5] (2007). URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=4211749&arnumber=4211979&count=305&index=229. Citations in this document: §2.
58. Sihan Qing, Tatsuaki Okamoto, Jianying Zhou (editors), *Information and communications security, third international conference, ICICS 2001, Xian, China, November 13–16, 2001*, Lecture Notes in Computer Science, 2229, Springer, 2001. ISBN 3-540-42880-1. See [8].
59. F. Rodríguez-Henríquez, Çetin Kaya Koç, *On fully parallel Karatsuba multipliers for GF(2^m)*, in [60] (2003), 405–410. Citations in this document: §2, §2.
60. S. Sahni (editor), *Proceedings of the international conference on computer science and technology*, Acta Press, 2003. See [59].
61. Arnold Schönhage, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik* 2, Acta Informatica **7** (1977), 395–398. ISSN 0001–5903. MR 55:9604. URL: http://cr.yp.to/bib/entries.html#1977/schoenhage. Citations in this document: §2.
62. Arnold Schönhage, Volker Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292. ISSN 0010–485X. MR 45:1431. URL: http://cr.yp.to/bib/entries.html#1971/schoenhage-mult. Citations in this document: §2.
63. William Stein (editor), *Sage Mathematics Software (Version 3.2.3)*, The Sage Group, 2009. URL: http://www.sagemath.org. Citations in this document: §1.
64. Andrei L. Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*, Soviet Mathematics Doklady **3** (1963), 714–716. ISSN 0197–6788. Citations in this document: §2.
65. Alfred J. van der Poorten, Andreas Stein (editors), *Algorithmic number theory, 8th international symposium, ANTS-VIII, Banff, Canada, May 17-22, 2008, proceedings*, Lecture Notes in Computer Science, 5011, Springer, 2008. ISBN 978-3-540-79455-4. See [22].
66. Vijay Varadharajan, Yi Mu (editors), *Information security and privacy—6th Australasian conference, ACISP 2001, Sydney, Australia, July 11-13, 2001, proceedings*, Lecture Notes in Computer Science, 2119, Springer, 2001. ISBN 978-3-540-42300-3. See [21].
67. Joachim von zur Gathen, Jamshid Shokrollahi, *Fast arithmetic for polynomials over* $\mathbf{F}_2$ *in hardware*, in [3] (2006), 107–111. Citations in this document: §2, §2, §2.
68. André Weimerskirch, Christof Paar, *Generalizations of the Karatsuba algorithm for efficient implementations* (2006). URL: http://eprint.iacr.org/2006/224. Citations in this document: §2.
69. Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin (editors), *9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings*, Lecture Notes in Computer Science, 3958, Springer, 2006. ISBN 978-3-540-33851-2. See [12], [25].
70. Jianying Zhou, Javier Lopez, Robert H. Deng, Feng Bao (editors), *Information security, 8th international conference, ISC 2005, Singapore, September 20–23, 2005, proceedings*, Lecture Notes in Computer Science, 3650, Springer, 2005. ISBN 3-540-29001-X. See [24].