Efficient Implementation of Elliptic Curve Point Operations Using Binary Edwards Curves

Richard Moloney*†,
School of Mathematical Sciences,
University College Dublin,
Ireland
richard.moloney@ucd.ie

Aidan O'Mahony,
Intel Shannon,
Co Clare,
Ireland
aidan.o.mahony@intel.com

Pierre Laurent,
Intel Shannon,
Co Clare,
Ireland
pierre.laurent@intel.com

Abstract

This paper presents a deterministic algorithm for converting points on an ordinary elliptic curve (defined over a field of characteristic 2) to points on a complete binary Edwards curve. This avoids the problem of choosing curve parameters at random. When implemented on a large (512 bit) hardware multiplier, computation of point multiplication using this algorithm performs significantly better, in terms of code complexity, code coverage and timing, than the standard implementation. In addition, we propose a simple modification to the birational equivalence detailed in the paper by Bernstein et al. which both reduces the number of inversions required in the affine mapping and has fewer exceptional points. Finally, we compare software implementations using this efficient point multiplication for binary Edwards curves with computations on elliptic curves in Weierstrass form.

*Research supported by Claude Shannon Institute, Science Foundation Ireland Grant 06/MI/006, and the Irish Research Council for Science, Engineering and Technology

1 Introduction

Binary Edwards curves (BECs) were introduced by Bernstein et al. in [1]. If k is a field of characteristic 2, $d_1,\ d_2 \in k$ with $d_1 \neq 0,\ d_2 \neq {d_1}^2 + d_1$, the binary Edwards curve with coefficients $d_1,\ d_2$ is the affine curve

$$E_{B,d_1,d_2} : d_1(x+y) + d_2(x^2+y^2) = (x+x^2)(y+y^2)$$

The addition law is given by $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \frac{d_1(x_1+x_2) + d_2(x_1+y_1)(x_2+y_2) + (x_1+{x_1}^2)(x_2(y_1+y_2+1)+y_1y_2)}{d_1 + (x_1+{x_1}^2)(x_2+y_2)}$$

$$y_3 = \frac{d_1(y_1+y_2) + d_2(x_1+y_1)(x_2+y_2) + (y_1+{y_1}^2)(y_2(x_1+x_2+1)+x_1x_2)}{d_1 + (y_1+{y_1}^2)(x_2+y_2)}$$

An ordinary elliptic curve E over k expressed in the form

$$v^2 + uv = u^3 + a_2u^2 + a_6$$

is birationally equivalent to a complete binary Edwards curve with coefficients $d_1,\ d_2$ where $\mathrm{Tr}(d_1) = \mathrm{Tr}(a_2)+1$, $\mathrm{Tr}(\sqrt{a_6}/{d_1}^2) = 1$ and $d_2 = {d_1}^2 + d_1 + \sqrt{a_6}/{d_1}^2$. Note that d_1 is generally not unique.

1.1 Review of some basic concepts

The trace function $\mathrm{Tr}:\mathbb{F}_{2^m} \to \mathbb{F}_2$ on a finite field of characteristic 2 is defined by [2]

$$\alpha \mapsto \sum_{i=0}^{m-1} \alpha^{2^i} = \alpha + \alpha^2 + \cdots + \alpha^{2^{m-1}}$$

Note that

$$\mathrm{Tr}(\alpha) = \mathrm{Tr}(\alpha^2) = \mathrm{Tr}(\sqrt{\alpha})$$

and

$$\mathrm{Tr}(\alpha+\beta) = \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta)$$

for all $\alpha,\ \beta \in \mathbb{F}_{2^m}$. If m is odd, $\mathrm{Tr}(1) = 1$. If α has minimal polynomial $x^t + a_{t-1}x^{t-1} + \cdots + 1$ over $\mathbb{F}_2$, then $\mathrm{Tr}(\alpha) = a_{t-1}$.

If m is odd, the half-trace function H:$\mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^m}$ is defined by [2]

$$\alpha \mapsto \sum_{i=0}^{(m-1)/2} \alpha^{2^{2i}} = \alpha + \alpha^4 + \alpha^{16} + \cdots + \alpha^{2^{m-1}}$$

Given $\alpha \in \mathbb{F}_{2^m}$, the square root of α is calculated as $\sqrt{\alpha} = \alpha^{2^{m-1}}$ [2].

1.2 Aim of this paper

Several points need to be made about the constraints of our physical system. We assume $k = \mathbb{F}_{2^m}$ where m is an odd prime (to avoid the Weil descent attack [2]). We assume E is an ordinary elliptic curve, denoted in short Weierstrass form, over k, and that at least one k-rational point of E, (u_1, v_1) is specified. Our purpose is to describe a method for obtaining the results of point operations on E by mapping E to a related complete binary Edwards curve, E_{B,d_1,d_2} (with $\mathrm{Tr}(d_2) = 1$).

Other constraints of our system are that the computation of the trace of an element is considered highly time–consuming, as is inversion of an element, and memory is assumed to be minimal, so storage of lists, vectors, etc. is not feasible. We do not assume access to a random number generator.

2 The birational equivalence

The birational equivalence from the binary Edwards curve E_{B,d_1,d_2} with coordinates (x, y) to a Weierstrass curve $v^2 + uv = u^3 + a_2u^2 + a_6$ with coordinates (u, v) is given by

$$u = \frac{d_1({d_1}^2 + d_1 + d_2)(x + y)}{(xy + d_1(x + y))}$$
$$v = d_1({d_1}^2 + d_1 + d_2)\left(\frac{(b + 1)x + by}{xy + d_1(x + y)} + d_1 + 1\right)$$

with one exceptional point $(0, 0)$ (identified with $\mathcal{O}$, the identity of the Weierstrass curve). We will make frequent use of b, denoting an element of k satisfying $b^2 + b = {d_1}^2 + d_2 + a_2$.

The inverse mapping is given by

$$x = \frac{d_1(u + {d_1}^2 + d_1 + d_2)}{(b + 1)u + v + ({d_1}^2 + d_1)({d_1}^2 + d_1 + d_2)}$$
$$y = \frac{d_1(u + {d_1}^2 + d_1 + d_2)}{bu + v + ({d_1}^2 + d_1)({d_1}^2 + d_1 + d_2)}$$

The exceptional points of this mapping are $\mathcal{O}$, $({d_1}^2+d_1+d_2, ({d_1}^2+d_1+d_2)({d_1}^2+d_1+b))$ and $({d_1}^2+d_1+d_2, ({d_1}^2+d_1+d_2)({d_1}^2+d_1+b+1))$. These formulae are obtained by simply composing the birational equivalence in section 2 of [1] with the isomorphism $v \mapsto v+bu$ which maps the curve $v^2+uv = u^3+({d_1}^2+d_2)u^2+a_6$ to $v^2+uv = u^3+a_2u^2+a_6$.

2.1 A modification of the birational equivalence

We present a modification of the birational equivalence from the Weierstrass curve to the complete binary Edwards curve which has only one exceptional point and reduces the number of inversions required. Define

$$z = ((b+1)u+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))(bu+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))$$

Then

$$x = \frac{d_1(u+{d_1}^2+d_1+d_2)(bu+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))}{z}$$
$$y = \frac{d_1(u+{d_1}^2+d_1+d_2)((b+1)u+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))}{z}$$

But

$$\begin{aligned} z &= (bu+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))^2+u(bu+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2)) \\ &= (b^2+b)u^2+v^2+uv+({d_1}^4+{d_1}^2)({d_1}^4+{d_1}^2+d_2^2)+u({d_1}^2+d_1)({d_1}^2+d_1+d_2) \end{aligned}$$

Using $b^2+b = {d_1}^2+d_2+a_2$, $v^2+uv = u^3+a_2u^2+a_6$ and $a_6 = ({d_1}^4+{d_1}^2+d_2^2)$ (from [1])

$$\begin{aligned} z &= ({d_1}^2+d_2)u^2+u^3+{d_1}^2({d_1}^4+{d_1}^2+d_2^2)+u({d_1}^2+d_1)({d_1}^2+d_1+d_2) \\ &= (u+{d_1}^2+d_1+d_2)(u^2+d_1u+{d_1}^2({d_1}^2+d_1+d_2)) \end{aligned}$$

Thus

$$x = \frac{d_1(bu+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))}{u^2+d_1u+{d_1}^2({d_1}^2+d_1+d_2)}$$
$$y = \frac{d_1((b+1)u+v+({d_1}^2+d_1)({d_1}^2+d_1+d_2))}{u^2+d_1u+{d_1}^2({d_1}^2+d_1+d_2)}$$

We claim that if E_{B,d_1,d_2} is a binary Edwards curve with $\mathrm{Tr}(d_2) = 1$ (shown in [1] to be a sufficient condition for completeness), this mapping has only one exceptional point, $\mathcal{O}$. For

$$u^2+d_1u+{d_1}^2({d_1}^2+d_1+d_2) = 0$$

to have a solution $u \in k$, we require that $\mathrm{Tr}\left(\frac{{d_1}^2({d_1}^2+d_1+d_2)}{{d_1}^2}\right) = \mathrm{Tr}({d_1}^2+d_1+d_2) = 0$, but $\mathrm{Tr}({d_1}^2+d_1+d_2) = \mathrm{Tr}(d_1)+\mathrm{Tr}(d_1)+\mathrm{Tr}(d_2) = 1$, by hypothesis.

3 Finding d_1

To compute point operations on an elliptic curve in Weierstrass form using a BEC, we need to find an appropriate d_1 parameter. The algorithms in this section carry this out in a deterministic manner (as opposed to choosing d_1 at random).

We assume the following are known:

m, a prime integer and p, an irreducible polynomial in $\mathbb{F}_2[x]$ of degree m, defining a field k. (We denote field elements as polynomials in x.)

a_2, $a_6 \in k$, $a_6 \neq 0$ defining an elliptic curve

$$E : v^2 + uv = u^3 + a_2u^2 + a_6$$

over k. We precompute $t = \mathrm{Tr}(a_2),\ r = \mathrm{Tr}(a_6)$

The desired output is a point (or set of points), the result of some sequence of point operations on E.

We need to find a $d_1 \in k$ such that $\mathrm{Tr}(d_1{+}a_2) = 1$, and $\mathrm{Tr}(\sqrt{a_6}/{d_1}^2) = 1$ (as required in [1]). We then define $d_2 = {d_1}^2 + d_1 + \sqrt{a_6}/{d_1}^2$, and find a b such that $b^2 + b = {d_1}^2 + d_2 + a_2$.

Observe that, using the properties listed in section 1.1, $\mathrm{Tr}(1) = 1$ and $\mathrm{Tr}(x)$, $\mathrm{Tr}(x^2)$,$\mathrm{Tr}(x^4)$, ... are known as the second coefficient of p. We denote $w = x{+}\mathrm{Tr}(x)$, noting that $\mathrm{Tr}(w) = 0$.

Algorithm 1 terminates with guaranteed success in a finite number of steps, except in the case $t = r = 0$. This case does not appear in any of the standards (e.g. NIST [3]) of which the authors are aware; Koblitz curves always have $r = \mathrm{Tr}(1) = 1$, and non-Koblitz curves are chosen such that they have a minimal cofactor of 2 (forcing $t = 1$, as per theorem 3.18 of [2]).

The other parameters of the mapping are d_2, which is directly calculated as $d_2 = {d_1}^2 + d_1 + \sqrt{a_6}/{d_1}^2$ and b, which satisfies $b^2 + b = {d_1}^2 + d_2 + a_2$ ($b = \mathrm{H}({d_1}^2 + d_2 + a_2)$). If we use algorithm 2 then the inversion used in computing d_2 disappears and is replaced with $d_2 = (d_1{+}e_1)^2{+}d_1$. It is also worth noting that the calculation of $1/(q^2{+}q{+}1)$, $(q^2{+}q{+}1)/q^2$ etc are actually trivial when q is chosen to be x and do not require use of the extended Euclidean algorithm[1].

[1] We extend the methods for modular inverse described in [4] to polynomials and fields of characteristic two. We believe this to be well known but are unable to find reference to this method.

Input: m, p, t, r, a_6, w
Postcondition: $\mathrm{Tr}(d_1) = \mathrm{Tr}(a_2) + 1$ and $\mathrm{Tr}(\sqrt{a_6}/{d_1}^2) = 1$
if $t = 0$ and $r = 1$ **then**
    Let $d_1 = 1$.
else
    if $t = 1$ and $r = 0$ **then**
        Let $d_1 = \sqrt[4]{a_6}$.
    else
        if $t = r = 1$ and $a_6 \neq 1$ **then**
            if $\mathrm{Tr}(1/(a_6 + 1)) = 1$ **then**
                Let $d_1 = \sqrt{a_6} + \sqrt[4]{a_6}$.
            else
                Let $d_1 = \sqrt[4]{a_6} + 1$.
        else
            if $t = 1$ and $a_6 = 1$ **then**
                if $\mathrm{Tr}(1/w) = 1$ **then**
                    Let $d_1 = w$.
                else
                    if $\mathrm{Tr}(1/(w + 1)) = 0$ **then**
                        Let $d_1 = 1/(w + 1)$.
                    else
                        Let $d_1 = 1 + 1/(w + 1)$.
            else
                if $t = r = 0$ **then**
                    if $\mathrm{Tr}(1/(a_6 + 1)) = 0$ **then**
                        Let $d_1 = \sqrt[4]{a_6} + 1$.
                    else
                        Let $i = 1$
                        Let $s = \sqrt{a_6}$
                        while $Tr({a_6}^{(2^i+1)}) = 0$ **do**
                            Let $s = s^2$
                            $i = i + 1$
                        Let $d_1 = 1/(s + 1)$

Algorithm 1: Algorithm to generate a suitable d_1

Input: a_6, a_2, *choose* $q = x$, f
Output: $d_1 = F, e_1 = E$
Let $A = \sqrt[4]{a_6}/(q^2 + q + 1)$
Let $B = q^2 A$
Let $C = qA$
if $tr(A) = tr(a_2) + 1$ **then**
    output$(d_1 = A, e_1 = (q^2 + q + 1))$
if $tr(B) = tr(a_2) + 1$ **then**
    output$(d_1 = B, e_1 = (q^2 + q + 1)/q^2)$
if $tr(A) + tr(B) = tr(a_2) + 1$ **then**
    output$(d_1 = A + B, e_1 = (q^2 + q + 1)/(q^2 + 1))$
if $tr(C) = 1$ **then**
    output$(d_1 = 1, e_1 = \sqrt[4]{a_6})$
else
    /* $Tr(a_2) = Tr(a_6) = 0$ */
    choose r with $0 < deg(r) < deg(f)$
    if $tr(1/(r + 1)) = 1$ **then**
        $D = r, E = 1/(r + 1$
    else
        $D = 1/r, E = r/(r + 1)$
    Let $dstart = D$
    Let $F = \sqrt[4]{a_6}(D + 1)$
    while $tr(F) = 0$ **do**
        Let $D = D^2, E = E^2$
        if $D = dstart$ **then**
            if $tr(1/r) = 1$ **then**
                $D = r + 1, E = 1/r$
            else
                $D = 1/(r + 1, E = (r + 1)/r$
        Let $F = \sqrt[4]{a_6}(D + 1)$

Algorithm 2: Alternative algorithm to generate a suitable d_1

4 Summary of procedure

We summarize the procedure used to carry out point operations on an ordinary elliptic curve over $\mathbb{F}_{2^m}$ using a complete binary Edwards curve.

Find d_1, d_2 and b as described in section 3.

Map the point (u, v) to a point$(x : y : z)$ on the projective binary Edwards curve:

$$\begin{aligned} x &= d_1(bu + v + ({d_1}^2 + d_1)({d_1}^2 + d_1 + d_2)) \\ y &= x + d_1 u \\ z &= u^2 + d_1 u + {d_1}^2({d_1}^2 + d_1 + d_2) \end{aligned}$$

Or, using $d_2 = {d_1}^2 + d_1 + \sqrt{a_6}/{d_1}^2$,

$$\begin{aligned} x &= d_1 bu + d_1 v + (d_1 + 1)\sqrt{a_6} \\ y &= x + d_1 u \\ z &= u^2 + d_1 u + \sqrt{a_6} \end{aligned}$$

Carry out point addition, doubling etc., in projective Edwards coordinates as described in [1]. Call the result $(x^{'} : y^{'} : z^{'})$

Map the resulting points $(x^{'} : y^{'} : z^{'})$ back to the points $(u^{'}, v^{'})$ on the affine Weierstrass-form elliptic curve:

$$\begin{aligned} u^{'} &= \sqrt{a_6}\left(\frac{(x^{'} + y^{'})z^{'}}{d_1 x^{'} y^{'} + {d_1}^2(x^{'} + y^{'})z^{'}}\right) \\ v^{'} &= \sqrt{a_6}\left(\frac{(b+1)x^{'}z^{'} + by^{'}z^{'}}{d_1 x^{'} y^{'} + {d_1}^2(x^{'} + y^{'})z^{'}} + 1 + \frac{1}{d_1}\right) \end{aligned}$$

5 Current Implementations

To fully understand the problem that is being solved by this approach, it is worthwhile examining existing implementations of elliptic curve cryptography (ECC) over GF(2^m). It is also important to consider that in order to develop a secure cryptosystem it is prudent to work alongside relevant standards such as [5]. We examine three implementations of ECC over GF(2^m).

5.1 Crypto++

Crypto++ [6] provides implementations of modular arithmetic, point arithmetic, ECDSA, ECDH, ECIES, ECNR and ECMQV over both GF(p) and GF(2^m). In the case of GF(2^m), the user is permitted to use elliptic curves from the FIPS standards or curves that the user may have selected themselves.

Internally, the basic point operations (point multiplication, point addition etc.) are based on points represented in affine coordinates on short Weierstrass curves only.

5.2 Miracl

Miracl [7] provides implementations of modular arithmetic and point arithmetic over both GF(p) and GF(2^m) and also supplies sample code for ECDH and ECDSA. In the case of GF(2^m), the user is permitted to use elliptic curves from the FIPS standards or curves the user may have created themself, however the field polynomial is restricted to either a pentanomial or trinomial.

Internally, the basic point operations are based on points represented in either affine coordinates or projective coordinates on short Weierstrass curves also.

5.3 A hardware implementation of ECC over a binary Edwards curve

The only implementation available that incorporates binary Edwards curves is found in [8]. In this thesis, the use of BECs is described and is implemented in the GEZEL hardware design language [9]. This implementation does not take into account the curves from the FIPS standards and therefore does not have the issue of mapping between short Weierstrass and binary Edwards curves.

6 Hardware

The hardware available to us is described in [10]. A simplified diagram of the hardware is shown in figure 1. This hardware consists of a very large multiplier, shifter, alu, memory (data and control), flags and FIFOs. As described in [10] section [0075], it is crucial for efficient operation to keep the pipeline of the multiplier full. This means that branches and testing operands are very costly as they break the multiplier pipeline.

7 Results

7.1 Our chosen implementation

For the purposes of comparing complexity and coverage metrics, we examined an implementation of elliptic curve operations using non-adjacent form (NAF) [11] [12] [13] as well as the right to left binary method for point multiplication described in [2] (algorithm 3.26). Implementation of NAF on our device proved impossible due to control store constraints until we were able to dramatically reduce instructions by using BECs.

7.2 Effect on Complexity

Code complexity can be considered a measure of the difficulty of providing an algorithm and is a common source of defects within source code [14]. One measure of complexity that is in common usage is that of "Cyclomatic Complexity" [15], which represents the complexity of an artifact by a number called a "McCabe number". It is recommended that for any given module, its McCabe complexity should not exceed 10 [14] (where a higher number indicates higher complexity). We use this measurement in our implementations of ECC. The tool we used in this paper to measure complexity is "CCCC" [16], and the results can be seen in table 1.

7.3 Effect on Coverage

Code coverage is a measure of how well an artifact of code is tested (and therefore gives some indication as to how reliable the code is [17]). We will concern ourselves mostly with block coverage and decision coverage. These terms "block coverage" and "decision coverage" are explained in some detail with examples in [18].

It seems natural that code related to security be 100% covered. Implementing ECC from the textbooks leaves us with incredibly complex code, which includes computationally intensive trace conditions. While there are many articles available detailing why complete code coverage can give false confidence, as [17] explains, we expect that as coverage increases, so does reliability. Therefore, it is our goal to reach 100% block and decision coverage.

Implementing the operations as described in [1] augmented with a deterministic binary Edwards curve coefficient generation described in this paper, the McCabe complexity decreases dramatically from **51** to **31**. If we choose only points on the curves detailed in [5] the complexity decreases even further to **28**. The comparisons can be seen in table 1.

Type	Coverage (in percent)	Complexity
Binary Edwards with NAF (all curves)	100	31
Binary Edwards with NAF (only curves from [5])	100	28
Without Binary Edwards and with NAF (all curves)	77	51

Table 1: Coverage and complexity comparison

Curve	Multiplications	Squarings	Additions
Binary Edwards	5	6	9
Weierstrass	8	3	4

Table 2: Comparison between cost of doubling on BEC and Weierstrass curves

7.4 Comparison between BEC and Weierstrass

It is worthwhile to compare the number of modular operations required to conduct a point addition and point doubling using both BECs and Weierstrass curves where the points are represented using projective coordinates. We further make the assumption that we do not know in advance that any of the z coordinates are 1 (to use this assumption we would need to implement two different versions of the addition and doubling). These figures come from the Explicit-Formulas Database (EFD)[19].

It is significant to note that on our hardware [10], an addition can execute in parallel with a multiplication. Therefore, it is reasonable to remove the additions from the below table. Also, a square operation is carried out by the multiplier. This means that the cost of a square is equal to that of a multiplication. This means we can compare the cost of doubling and addition in terms of (number operations = number of squares + number of multiplications).

We then see that the cost of a point double on our device using BECs is 11 multiplications versus Weierstrass which also needs 11 multiplications. However, the cost from the EFD does not take into account the cost of checking that if the operations is attempting to double the point at infinity.

It is preferable, therefore, to double using the BEC method. The same logic can be applied to the addition which required even more checks in code before the addition itself can be carried out. Table 2 and table 3 show the multiplication cost between the two addition methods and the two doubling methods.

Curve	Multiplications	Squarings	Additions
Binary Edwards	25	1	15
Weierstrass	15	1	7

Table 3: Comparison between cost of addition on BEC and Weierstrass curves

Curve	Operation	Weierstrass (ms per op)	Binary Edwards (ms per op)
B-163	ECDH offline, with precomputation	0.22	0.26
B-163	ECDSA Verify	1.18	1.45
B-233	ECDH offline, with precomputation	0.36	0.43
B-233	ECDSA Verify	1.85	2.27
B-283	ECDH offline, with precomputation	0.51	0.65
B-283	ECDSA Verify	2.64	3.44
B-571	ECDH offline, with precomputation	1.79	2.60
B-571	ECDSA Verify	9.62	13.12

Table 4: Performance of Weierstrass and binary Edwards operations on i7

7.5 Comparison between BEC and Weierstrass on IA

As a further comparison, we integrated BECs into the cryptography library "Miracl" [7]. The results are in table 2 and were carried out on a Intel(R) Core(TM) i7 920 CPU 2.67GHz eight core 64-bit machine with 6GB RAM. The tests were carried out using the "bmark" program supplied with "Miracl".

As we can see from table 4, on a standard IA the binary Edwards curves do not perform as well as the Weierstrass curves. However, it is still important to remember that the coverage for the binary Edwards implementation is still considerably higher and less complex then the alternative. Also, we shall see in the next section how the performance is significantly different if implemented on a different processor.

7.6 Comparison between BECs and Weierstrass on EP80579

Our device, which is a member of the Intel(R) EP80579 Integrated Processor product line, accelerates certain ellipic curve operations and acceleration is offloaded to a cryptographic engine (Intel(R) QuickAssist Technology) similar to that described in [10]. We are able to take advantage of better pipelining of instructions (allowing us to parallelize the similar trace, half trace and modular square root operations over characteristic two) and a dedicated polynomial multiplier. It is also important to note that the computation of a trace or half trace or square root cost less then computing the extended Euclidean algorithm. These

Curve	Cycles	Number of instructions
Binary Edwards	1859356	378
Weierstrass	2405558	385

Table 5: Cycle comparison between BEC and Weierstrass point multiplication on a member of the EP80579 family of embedded processors

advantages, used together with BECs actually make a difference in performance when compared to a Weierstrass implementation. Due to the fact that we do not have to break the pipeline with checks for the point of infinity, and the reduced control store allowed us to integrate non-adjacent form, we are able to increase the performance of BECs such that it is approximately **25%** faster than the equivalent Weierstrass version. Table 5 gives a cycle comparison between a point multiplication using BECs and a point multiplication using Weierstrass curves (point multiplication is $[2^{209} + 2^{87}]$P on NIST curve K-409).

8 Future Work

We are investigating the possibility that the selection of d_1 might come from the solution of some quadratic equation. We are also attempting to find d_1, d_2 pairs such that d_2/d_1 is optimised for faster computations on the curves listed in the NIST standards [3]. This would be useful to implementers of ECC.

9 Acknowledgements

The authors would like to thank Catriona Lucey, Gary McGuire and other reviewers for taking the time to review this paper and providing helpful comments.

10 Conclusion

There is significant benefit to using BECs as is demonstrated by the reduced complexity (McCabe complexity decreases dramatically from **51** to **31**) and increased test coverage (from **77%** to **100%**). These factors alone are of huge importance to implementors of cryptographic algorithms. We have also seen that, while on an Intel(R) IA32 processor there is some minimal speed impact, if implemented with a large multiplier on a different architecture there is a **25%** performance increase. Finally, our work has shown that there exists a modified birational equivalence with fewer exceptional points, and that it is possible

for us to deterministically map points from Weierstrass representation to binary Edwards representation with minimal effort.

11 Figures

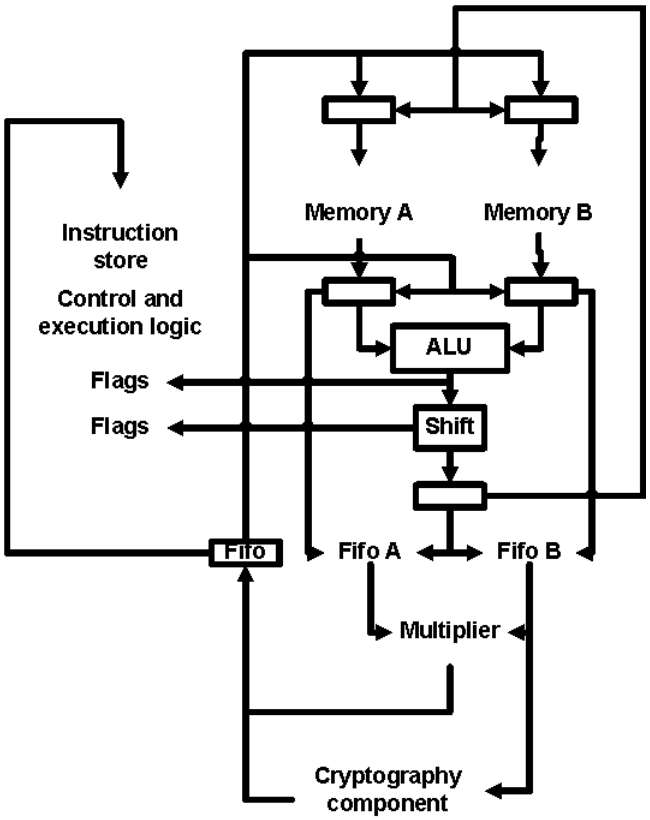

Figure 1: Cryptographic processor

References

[1] D. J. Bernstein, T. Lange, and R. Rezaeian Farashahi, "Binary edwards curves," in *CHES '08: Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems*, (Berlin, Heidelberg), pp. 244–265, Springer-Verlag, 2008.

[2] S. V. D. Hankerson, A. Menezes, *Guide To Elliptic Curve Cryptography.* Springer, 2004.

[3] N. I. of Standards and Technology, "Recommended elliptic curves for federal government use." `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`, 1999.

[4] H. S. Warren, *Hacker's Delight*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] NIST, "Fips 186-3." `http://csrc.nist.gov/publications/PubsDrafts.html`.

[6] NSA, "Crypto++ cryptographic library 5.6.0." `http://www.cryptopp.com/`.

[7] M. Scott, "Miracl." `http://www.shamus.ie`.

[8] U. Kocabas, "Hardware Implementations Of ECC Over A Binary Edwards Curve," Master's thesis, Katholieke Universiteit Leuven, Belgium, 2009.

[9] "Gezel 2.4." `http://rijndael.ece.vt.edu/gezel2/index.php/Main_Page`.

[10] W. K. Feghali, W. C. Hasenplaugh, G. M. Wolrich, D. F. Cutter, V. Gopal, and G. Gaubatz, "Multiplier european patent application ep1966680." `http://www.freepatentsonline.com/EP1966680A2.html`, September 2008.

[11] K. Okeya, K. Schmidt-samoa, C. Spahn, and T. Takagi, "Signed binary representations revisited," in *Advances in Cryptology CRYPTO 2004, Lecture Notes in Computer Science 3152 (2004), 123139. 151*, pp. 123–139, Springer, 2004.

[12] B. Qin, M. Li, F. Kong, and D. Li, "New left-to-right minimal weight signed-digit radix-r representation," *Computers & Electrical Engineering*, vol. 35, no. 1, pp. 150 – 158, 2009.

[13] M. Joye and S.-M. Yen, "New minimal modified radix-r representation with applications to smart cards," in *PKC '02: Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, (London, UK), pp. 375–384, Springer-Verlag, 2002.

[14] NIST, "Nist structured testing methodology." `http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/sttoc.htm`.

[15] T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, pp. 308–320, Dec. 1976.

[16] T. Littlefair, "C and c++ code counter." `http://sourceforge.net/projects/cccc`.

[17] F. Del Frate, P. Garg, A. Mathur, and A. Pasquini, "On the correlation between code coverage and software reliability," in *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pp. 124–132, Oct 1995.

[18] J. Horgan and A. Mathur, "Assessing testing tools in research and education," *Software, IEEE*, vol. 9, pp. 61–69, May 1992.

[19] D. J. Bernstein and T. Lange, "Explicit-formulas database." `http://www.hyperelliptic.org/EFD/index.html`.