# The Hessian Form of an Elliptic Curve

N.P. Smart

Dept. Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB
nigel@cs.bris.ac.uk

**Abstract.** In this paper we use the Hessian form of an elliptic curve and show that it offers some performance advantages over the standard representation. In particular when a processor allows the evaluation of a number of field multiplications in parallel (either via separate ALU's, a SIMD type operation or a pipelined multiplication unit) one can obtain a performance advantage of around forty percent.

## 1 Introduction

Much research has been conducted on implementing cryptographic operations on various types of computer architectures. For example standard superscalar RISC and CISC processors [3] [6], smart cards [9] and FPGAs [8] . In this paper we are interested in producing highly efficient implementations of operations needed in elliptic curve cryptography by exploiting parallelism.

In [3] the instruction level parallelism (ILP) in the various AES candidates was examined. This is an important area of research for any cryptographic algorithm as most algorithms will be implemented on superscalar processors which make use of ILP to increase performance. In most cryptographic algorithms the basic blocks are dependent, hence the only natural level of parallelism is at the instruction level. We shall show in this paper that this is not true for elliptic curve cryptosystems. We shall show that for elliptic curve systems one can make use of parallelism at the basic block, or function, level. We shall then go on to show that a special class of elliptic curves, namely those with a point of order three, have addition laws which are highly parallel. In addition the curves will be able to be used in both even and large characteristic. We shall give a comparison of a standard elliptic curve representation against the Hessian form on an FPGA.

We see significant advantages for using the ideas in this paper when we consider the use of custom processors for cryptographic operations. These are common at both the high and low end of the markets: Smart cards require specialized cryptographic coprocessors to be able to perform a single cryptographic operation in a reasonable amount of time, whilst high end servers often require cryptographic accelerator boards to as to increase the throughput of the overall traffic.

We envisage a processor which has the ability to conduct a number of multiprecision arithmetic operations in parallel. We assume that the field is either

$\mathbb{F}_p$ or $\mathbb{F}_{2^n}$, as is common in elliptic curve systems. This can be implemented in a number of ways, either as a SIMD operation, via a number of standard coprocessors operating in parallel, by having a number of Arithmetic Logic Units (ALUs) operating in parallel or by having a single pipelined multiplication unit. To aid the discussion later we shall assume up to three field operations can be applied in parallel in a SIMD fashion, i.e. at most three multiplications or three additions. It will turn out that performing additions in parallel is not as important, and in any case can be done virtually for free in characteristic two.

## 2 Cryptographic Algorithms

It is clear that standard implementations of RSA and discrete logarithm type systems cannot make use of the architecture proposed in the previous section. Of course they could perform a number of RSA exponentiations in parallel, but we would not obtain a performance advantage for a single modular exponentiation. The reason for this lack of improvement is that each modular multiplication required in a group exponentiation algorithm is dependent on the previous operations.

For elliptic curve cryptography (ECC) systems the situation is very different. In ECC one trades smaller sizes for the field elements against a more complicated group operation. We have already remarked that the group exponentiation techniques do not offer any opportunity for parallelism. However, the group operation itself does offer such opportunities.

To see this consider the following description of the doubling formulae for a point $P = (X, Y)$ on a curve over a field of characteristic $p > 3$ given by

$$Y^2 = X^3 - 3X + b.$$

We assume the point is given in Jacobian projective coordinates $P = (x/z^2, y/z^3)$ and the output is given by $(x', y', z')$;

$$\begin{array}{lll}
\lambda_1 = x^2 & \lambda_2 = z^2 & \lambda_3 = y^2 \\
\lambda_4 = yz & \lambda_5 = x\lambda_3 & \lambda_6 = \lambda_3^2 \\
\lambda_7 = \lambda_2^2 & z' = 2\lambda_4 & \lambda_8 = 4\lambda_5 \\
\lambda_9 = \lambda_1 - \lambda_7 & & \\
\lambda_{10} = 8\lambda_6 & \lambda_{11} = 3\lambda_9 & \lambda_{12} = 2\lambda_8 \\
\lambda_{13} = \lambda_{11}^2 & & \\
x' = \lambda_{13} - \lambda_{12} & & \\
\lambda_{14} = \lambda_8 - x' & & \\
\lambda_{15} = \lambda_{11}\lambda_{14} & & \\
y' = \lambda_{15} - \lambda_{10} & &
\end{array}$$

Each row corresponds to the (at most) three operations which can be carried out in parallel. As one can see one can obtain a limited improvement in performance by exploiting the parallel nature of the computation. However, this is rather restricted due to our SIMD based constraint of not allowing the mixing of field additions and multiplications. Even if we dropped this constraint the above algorithm would not improve performance that much. If we assume each row in the above table can be executed in the same time on a SIMD/pipelined style processor as a single operation could on a standard processor, we would obtain roughly a 50 percent performance improvement.

Similar considerations apply to all the other standard algorithms for addition and doubling on elliptic curves in even and large prime characteristic. There is generally a set of operations at the beginning of each operation which lend themselves to parallel execution followed by a sequence of operations which are highly dependent. Overall one can obtain roughly a 50–60 percent improvement in performance.

In the next section we present a special type of elliptic curve which allows one to obtain a five fold increase in performance over the standard sequential point addition algorithm and a three fold increase over the standard sequential point doubling algorithm.

## 3 The Hessian Form of an Elliptic Curve

Let $k = \mathbb{F}_q$ denote a finite field with $q$ a prime power such that $q \equiv 2 \pmod 3$, we include the case of characteristic two fields. Let $E$ denote an elliptic curve over $k$ which has a $k$-rational point of order 3. The restriction on the choice of $k$ is for two reasons; Firstly it implies that although we have a point of order 3 we only have two of them rather than a full set of eight. Secondly it implies that the construction below is guaranteed to apply.

Note that this constraint on the field and curve means that the recommended curves specified in the ANSI and FIPS standards cannot be made into Hessian form. However, these constraints are consistent with the recommendations for curve parameters specified in the IEEE, ANSI and SECG standards. So the Hessian form is still able to be used in standard compliant implementations, one just cannot use the recommended curves contained in some standards.

By moving a point of order three to the origin, we can assume our elliptic curve has the form

$$E : Y^2 + a_1XY + a_3Y = X^3.$$

This curve has discriminant

$$\Delta = a_3^3(a_1^3 - 27a_3) = a_3^3\delta.$$

The points of order three on $E$ are given by $(0,0)$ and $(0,-a_3)$.

We wish to find a more convenient model for our elliptic curve $E$. To do this we perform the following transformations: We let $\mu$ denote a root of the polynomial

$$T^3 - \delta T^2 + \delta^2 T/3 + a_3\delta^2 = 0.$$

Since $q \equiv 2 \pmod 3$ every element in $k$ has a unique cube root and we can determine $\mu$ from the formula

$$\mu = \frac{1}{3}\left(\left(-27a_3\delta^2 - \delta^3\right)^{1/3} + \delta\right).$$

Now define

$$D = 3\frac{\mu - \delta}{\mu}.$$

The curve $E$ is then birationally equivalent to the curve

$$C : x^3 + y^3 + z^3 = Dxyz,$$

which is called the cubic Hessian form [2]. The change of variable to pass from $E$ to $C$ is given by

$$\begin{aligned} x &= \frac{a_1(2\mu - \delta)}{3\mu - \delta}X + Y + a_3, \\ y &= \frac{-a_1\mu}{3\mu - \delta}X - Y, \\ z &= \frac{-a_1\mu}{3\mu - \delta}X - a_3. \end{aligned}$$

## 3.1 Example Curve

Our example curve will be in characteristic two, defined over the field of $2^{191}$ elements. We shall represent the field by a polynomial basis in $t$ over $\mathbb{F}_2$, where

$$t^{191} + t^9 + 1 = 0.$$

Elements of $k$ we will represent by hexadecimal numbers, prefixed by the string `0x`, which correspond to the associated polynomial being thought of as a polynomial over the integers and then evaluated at 2. For example `0x11` corresponds to the polynomial $x^4 + 1$.

Consider the elliptic curve given by

$$E' : y^2 + xy = x^3 + x^2 + b$$

where

$$b = \texttt{0x4DE3965E00F2A1C6C9750156A6FEFBE5EEF780BF3EF20E48}.$$

This curve has group order

$$6 \cdot q = 3138550867693340381917894711648254768837315541933943803842,$$

where $q$ is a prime. A point of order 3 on $E'$ is given by the point $(x_3, y_3)$, where

$$\begin{aligned} x_3 &= \texttt{0x4763CFBC4340674B749E57887850E92C9B6BEDF58EEDC3BF}, \\ y_3 &= \texttt{0x14A96A1E53DCC3E73CFB22B80E8658CE0D6D8E82ED2AEC7D}. \end{aligned}$$

If we perform the following change of variable

$$X = x + x_3, \qquad Y = y + sx + x_3^2,$$

where

$$s = \frac{x_3^2 + y_3}{x_3},$$

then we obtain an elliptic curve $E$ in the form

$$E : Y^2 + XY + x_3 Y = X^3.$$

We can now use the above transformation to determine that the Hessian form $C$ is given by

$$x^3 + y^3 + z^3 = Dxyz,$$

where

$$D = \texttt{0x16A4C7C2030FAD1380ABF8C2D47DC3E0C20AF62F6EDD06A7}.$$

A point of order $q$ on $C$ is given by $(x, y, z)$, where

$$\begin{aligned} x &= \texttt{0x52FD0CE78D0651B4F66D2F4E12E170CA3E429F6A06433B22}, \\ y &= \texttt{0x1BECA50368403F3D13173968082B035397C77830A9D90E5D}, \\ z &= \texttt{0x2B08F7C0CCAC86151AA6FECABDD2D052BD60924F28A6A78E}. \end{aligned}$$

## 4 The Hessian Group Law

The group law on curves in Hessian form has been known to be particularly simple for a long time, see [1, Formulary], [2] and [5]. The zero of the group law on $C$ is given by $(1, -1, 0)$. The two points of order three are given by $(0, 1, -1)$ and $(1, 0, -1)$. If $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$, we define

$$\begin{aligned} -P &= (y_1, x_1, z_1), \\ P + Q &= (x_3, y_3, z_3), \end{aligned}$$

where

$$\begin{aligned} x_3 &= y_1^2 x_2 z_2 - y_2^2 x_1 z_1, \\ y_3 &= x_1^2 y_2 z_2 - x_2^2 y_1 z_1, \\ z_3 &= z_1^2 y_2 x_2 - z_2^2 y_1 x_1. \end{aligned}$$

The formulae for point doubling are given by $[2]P = (x_2, y_2, z_2)$ where

$$\begin{aligned} x_2 &= y_1(z_1^3 - x_1^3), \\ y_2 &= x_1(y_1^3 - z_1^3), \\ z_3 &= z_1(x_1^3 - y_1^3). \end{aligned}$$

These formulae apply both in even and large prime characteristic. The addition formulae requires 12 field multiplications, whilst the doubling formulae requires 6 field multiplications and three squarings.

We first compare the sequential operation of the above group law with the usual formulae.

For curves over fields of large prime characteristic, given by the usual model, it is best to use a mixed coordinate representation of points, see [4]. In the most common case of using Jacobian projective coordinates for doubling and mixed Jacobian/affine coordinates for addition we can perform an addition in 8 multiplications and 3 squarings. A doubling will take 4 multiplications and 4 squarings.

In fields of characteristic two, again in the standard model, we can again use a mixed coordinate system, but this time using the projective coordinates proposed in [7]. One can now perform an addition using 9 multiplications and 4 squarings. A doubling can be performed in 4 multiplications and 5 squarings.

So for sequential execution of the multiplication operations the standard representation appears to be more efficient. However, the group operations for the Hessian form can be performed in a highly parallel way. As before we assume that at most three multiprecision multiplications can be performed in parallel.

The addition of two points, $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$, in the Hessian form can be carried out as follows

$$\begin{array}{lll}
\lambda_1 = y_1 x_2 & \lambda_2 = x_1 y_2 & \lambda_3 = x_1 z_2 \\
\lambda_4 = z_1 x_2 & \lambda_5 = z_1 y_2 & \lambda_6 = z_2 y_1 \\
s_1 = \lambda_1 \lambda_6 & s_2 = \lambda_2 \lambda_3 & s_3 = \lambda_5 \lambda_4 \\
t_1 = \lambda_2 \lambda_5 & t_2 = \lambda_1 \lambda_4 & t_3 = \lambda_6 \lambda_3 \\
x_3 = s_1 - t_1 & y_3 = s_2 - t_2 & z_3 = s_3 - t_3
\end{array}$$

The case of the two projective points being equivalent, and the doubling operation needing to be called, can be detected from the condition

$$\lambda_1 = \lambda_2 \text{ and } \lambda_3 = \lambda_4.$$

The point doubling operation on the Hessian can also be expressed in a similar highly parallel way

$$\begin{array}{lll}
\lambda_1 = x_1^2 & \lambda_2 = y_1^2 & \lambda_3 = z_1^2 \\
\lambda_4 = x_1 \lambda_1 & \lambda_5 = y_1 \lambda_2 & \lambda_6 = z_1 \lambda_3 \\
\lambda_7 = \lambda_5 - \lambda_6 & \lambda_8 = \lambda_6 - \lambda_4 & \lambda_9 = \lambda_4 - \lambda_5 \\
x_2 = y_1 \lambda_8 & y_2 = x_1 \lambda_7 & z_2 = z_1 \lambda_9.
\end{array}$$

## 5 FPGA Implementation

We implemented the example curve above both using the standard representation and the Hessian form on a Xilinx4000XL FPGA. The code was written using the HandelC language and compiler produced by Embedded Solutions Ltd. This converts a C like language into a netlist which is then passed through Xilinx tools to produce the final FPGA bitmap.

The implementation was made to test the relative performance of the two models and not to maximize the speed. The FPGA implemented a simple point multiplication algorithm; taking as input a 191 bit integer $m$ and returning the projective point corresponding to $[m]P$, where $P$ is the point of order $q$ on the appropriate models given earlier.

The point multiplication used was the simple right-to-left binary method. The code for both representations made as much use of the parallel nature of the double and add routines as could be accommodated.

The following timings were obtained as an average over a number of calls to the FPGA;

| Form of Curve | Point Multiplication Time |
|---|---|
| Standard (Sequential) | 77.238 ms |
| Standard(Parallel) | 17.711 ms |
| Hessian | 11.821 ms |

So we see that the Hessian form gives around a 40 percent performance improvement over the standard form, even when we exploit all the inherent parallelism in the standard formulae.

## 6 Conclusion

We have shown that using the Hessian form of an elliptic curve allows us to implement the point addition and point doubling operation in a highly parallel way. This can be exploited by a number of processor architectures such as those which have a SIMD style instruction set, those which have multiple ALU's or those which have a pipelined finite field multiplier.

We have shown, by implementing a demonstration example on an FPGA in characteristic two, that the Hessian form does in fact give a significant performance improvement in real life.

## References

1. J.W.S. Cassels. *Lectures on Elliptic Curves.* LMS Student Texts, Cambridge University Press, 1991.
2. D.V. Chudnovsky and G.V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorisation tests. *Adv. in Appl. Math.*, **7**, 385–434, 1987.
3. C. Clapp. Instruction level parallelism in AES Candidates. *Second Advanced Encryption Standard Candidate Conference*, Rome March 1999.
4. H. Cohen, A. Miyaji and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Advances in Cryptology, ASIACRYPT 98.* Springer-Verlag, LNCS 1514, 51–65, 1998.
5. M. Desboves. Résolution en nombres entiers et sous sa forme la plus générale, de l'équation cubique, homogène, à trois inconnues. *Nouvelles Ann. de Math.*, **45**, 545-579, 1886.
6. C.K. Koc, T. Acer and B.S. Kaliski Jnr. Analyzing and comparing Montgomery multiplication algorithm. *IEEE Micro*, **16**, 26–33, June 1996.

7. J. López and R. Dahab. Improved algorithms for elliptic curve arithmetic in $GF(2^n)$ In *Selected Areas in Cryptography - SAC '98*, Springer-Verlag, LNCS 1556, 201–212, 1999.
8. G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for $GF(2^m)$. In *Cryptographic Hardware and Embedded Systems (CHES) 2000*, Springer-Verlag, LNCS 1965, 41–56, 2000.
9. A.D. Woodbury, D.V. Bailey and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *Smart Card and Advanced Applications, CARDIS 2000*, 71–92, Kluwer, 2000.