An FPGA Implementation of an Elliptic Curve Processor over $GF(2^m)$

Nele Mentens*
K.U.Leuven, ESAT/COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
Tel: +32 16 32 18 85
Fax: +32 16 32 19 69
Nele.Mentens@esat.kuleuven.ac.be

Sıddıka Berna Örs
K.U.Leuven, ESAT/COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
Tel: +32 16 32 18 85
Fax: +32 16 32 19 69
Siddika.BernaOrs@esat.kuleuven.ac.be

Bart Preneel
K.U.Leuven, ESAT/COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
Tel: +32 16 32 11 48
Fax: +32 16 32 19 69
Bart.Preneel@esat.kuleuven.ac.be

ABSTRACT

This paper describes a hardware implementation of an arithmetic processor which is efficient for elliptic curve (EC) cryptosystems, which are becoming increasingly popular as an alternative for public key cryptosystems based on factoring. The modular multiplication is implemented using a Montgomery modular multiplication in a systolic array architecture, which has the advantage that the clock frequency becomes independent of the bit length m.

Categories and Subject Descriptors: B.2.1 [Hardware]: Arithmetic and Logic Structures—*Design Styles*

General Terms: Design

Keywords: Elliptic Curve Cryptosystems, FPGA, Montgomery Modular Multiplication

1. INTRODUCTION

Elliptic curve (EC) cryptography was proposed independently by Miller [1] and Koblitz [2] in the 1980's. The benefits of EC, when compared with classical cryptosystems such as RSA [3] include: higher speed, lower power consumption and smaller certificates, which are particularly useful for wireless applications.

The performance of an EC cryptosystem and of several other public key cryptosystems strongly depends on the implementation of finite field arithmetic. In this article a hardware architecture of a processor for an EC cryptosystem over the finite field $GF(2^m)$ is presented.

*Nele Mentens and Sıddıka Berna Örs are funded by research grants of the Katholieke Universiteit Leuven, Belgium. This work was supported by Concerted Research Action GOA-MEFISTO-666 of the Flemish Government and by the FWO "Identification and Cryptography" project (G.0141.03).

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

The remainder of this paper is organized as follows. In Section 2 we discuss the related work. Section 3 provides the necessary mathematical background to understand Section 4, that describes the hardware implementation. Section 5 presents some implementation results and Section 6 concludes the paper.

2. PREVIOUS WORK

Agnew *et al.* reported the first results for performing elliptic curve operations over $GF(2^{155})$ on hardware [4]. A detailed overview of other EC cryptosystems over $GF(2^m)$ is given in [5]. All these previous implementations use a normal modular multiplication to perform the field multiplication in $GF(2^m)$. Our work uses MMM in a systolic array architecture, which allows to make the clock frequency independent of the bit length m.

3. MATHEMATICAL BACKGROUND

3.1 Elliptic curves over $GF(2^m)$

An elliptic curve E is often expressed in terms of the Weierstrass equation:

$$y^2 + xy = x^3 + ax^2 + b\,, \tag{1}$$

where $a, b, x, y \in GF(2^m)$, $b \neq 0$.

The points on the curve and the point at infinity $\mathcal{O}$ form an abelian group together with the operation of "addition". The point or scalar multiplication, this is the multiplication of a point on the curve with a scalar, is the main operation for EC cryptography. This operation can be calculated by using the double-and-add algorithm shown in Algorithm 1. For details see [1, 2, 6, 7].

Algorithm 1 Elliptic Curve Point Multiplication

Require: EC point $P = (x, y)$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$, $k_{l-1} = 1$
Ensure: $Q = (x', y') = kP$
1: $Q \leftarrow P$
2: **for** i from $l-2$ downto 0 **do**
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **if** $k_i = 1$ **then**
5: $\quad\quad Q \leftarrow Q + P$
6: $\quad$ **end if**
7: **end for**

In the above definition of the EC group affine coordinates are used, but so-called projective coordinates have some im-

plementation advantages. In affine coordinates, the point addition uses many field inversions. A field inversion is a very expensive operation. In projective coordinates, only one inversion needs to be performed at the end of a point multiplication operation. The benefits of using projective coordinates in EC cryptosystems are explained in more detail in [6].

3.2 Projective coordinates

A *projective plane*, denoted by P^2, is defined to be the set of equivalence classes of triples (X, Y, Z), not all zero. (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are said to be equivalent if there exists a $\lambda \neq 0 \in GF(2^m)$ such that $X_1 = \lambda X_2$, $Y_1 = \lambda^2 Y_2$ and $Z_1 = \lambda Z_2$. Each equivalence class is called a projective point $(x, y, 1)$, where $x = X/Z$ and $y = Y/Z^2$. The *projective equation* derived from the affine equation Eq. (1) is given by $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$.

To convert an affine point (x, y) to a projective point, one sets $X = x$, $Y = y$, $Z = 1$. To convert a projective point (X, Y, Z) to an affine point, we compute $x = X/Z$, $y = Y/Z^2$. The formulae for elliptic curve point addition and doubling are given by López and Dahab in [8].

3.3 Polynomial representation

Our algorithm for Montgomery modular multiplication is defined on the polynomial representation. An element a of $GF(2^m)$ is then represented by a polynomial with m coefficients, as follows:

$$a(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0\,, \qquad (2)$$

where the coefficients $a_i \in GF(2)$. In the word-level description of the algorithms, these bits are grouped into s words of equal length. Let w be the word length and $m = s \cdot w$. Hence, a can be written as $a(x) = \sum_{i=0}^{s-1} A_i(x)x^{iw}$, where each polynomial $A_i(x)$ corresponds to a word of length w, or $A_i(x) = a_{iw+w-1}x^{w-1} + \cdots + a_{iw+1}x + a_{iw}$.

The addition of two elements a and b in $GF(2^m)$ is performed by adding the polynomials $a(x)$ and $b(x)$, where the coefficients are added in the field $GF(2)$. This is equivalent to a bit-wise XOR operation on the vectors a and b. In order to multiply two elements a and b in $GF(2^m)$, we need to select an irreducible polynomial of degree m. Note that a different choice of polynomial leads to a different finite field representation, but all finite fields with the same number of elements are isomorphic. Different choices of irreducible polynomials are discussed in [9]. Let $p(x)$ be an irreducible polynomial of degree m over the field $GF(2)$, hence $p_m = 1$ and $p_0 = 1$. The product $c = a \cdot b$ in $GF(2^m)$ is obtained by computing $c(x) = a(x)b(x) \bmod p(x)$, where $c(x)$ is a polynomial of length m, representing the element $c \in GF(2^m)$.

3.4 Montgomery modular multiplication over $GF(2^m)$

The Montgomery multiplication of a and b is defined as $c(x) = a(x)b(x)r^{-1}(x) \bmod p(x)$ where $r(x) = x^m \bmod p(x) = (p_{m-1}p_{m-2}\cdots p_1 p_0)$ [10]. This equation can be evaluated with Algorithm 2 given by Koç and Acar in [11].

Algorithm 2 requires the computation of the polynomial $P_0'(x) = P_0^{-1}(x) \bmod x^w$, that consists of w bits, where w is the word length.

4. HARDWARE IMPLEMENTATION

Our Elliptic Curve processor (ECP) can be divided into four hierarchical levels as shown in Fig. 1.

Algorithm 2 Word-Level Montgomery modular multiplication

Require: $a(x)$, $b(x)$, $p(x)$, $P_0'(x)$
Ensure: $c(x) = a(x)b(x)x^{-m} \bmod p(x)$
1: $c(x) = 0$
2: **for** i from 0 to $(s-1)$ **do**
3: $\quad M(x) = (C_0(x) + A_i(x)B_0(x))\, P_0'(x) \bmod x^w$
4: $\quad c(x) = (c(x) + A_i(x)b(x) + M(x)p(x))\,/x^w$
5: **end for**

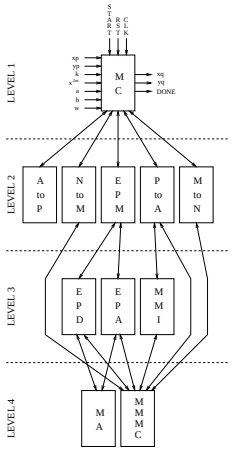

Figure 1: EC point multiplier circuit block diagram

The operation blocks at each level from top to bottom are as follows:

- **Level 1:** Main Controller (MC)
- **Level 2:**
  1. affine to projective converter (AtoP) (4.4)
  2. normal to Montgomery converter (NtoM) (4.5)
  3. EC point multiplier (EPM) (4.6)
  4. projective to affine converter (PtoA) (4.7)
  5. Montgomery to normal converter (MtoN) (4.8)
- **Level 3:**
  1. EC point doubling circuit (EPD) (4.2)
  2. EC point addition circuit (EPA) (4.2)
  3. modular multiplicative inverter (MMI) (4.3)
- **Level 4:**
  1. modular addition (MA) (4.1)
  2. Montgomery modular multiplication circuit (MMMC) (4.1)

The bit length m, the word length w and the key length l are input parameters to the circuit.

4.1 Modular addition and Montgomery modular multiplication circuit

The Modular Addition Circuit in $GF(2^m)$ is a bitwise logical XOR operation. The output is clocked into a register, so the computation is finished in one clock cycle.

The MMMC that performs Algorithm 2 is designed as a systolic array with variable bit length m and variable word length w. This systolic array architecture makes the maximum clock frequency independent of the bit length m. The clock frequency only depends on the word length w. The details of the MMMC can be found in [12].

As explained in Section 3.4 $P_0'(x) = P_0^{-1}(x) \bmod x^w$ needs to be computed. This is done by using the observation that $P_0(x)$ and its inverse satisfy $P_0(x)P_0^{-1}(x) = 1 \bmod x^i$ for $i = 1, 2, \cdots w$ [11]. We have designed a circuit that calculates the coefficients of the polynomial P_0^{-1} during the first clock cycle and writes the result in a register. Then the value of this register is used as an input for the MMMC in Algorithm 2.

4.2 EC point doubling and addition

Algorithm 3 realises an EC point doubling (a) and an EC point addition (b) based on [8]. The inputs to the algorithms are points on the curve that are written in projective coordinates, as explained in Section 3.2. The output points are also in projective coordinates.

The doubling algorithm consists of eleven steps and needs three temporary registers. In each step a modular addition and/or a Montgomery modular multiplication is executed. Because MA and MMMC are seperate hardware blocks, the operations can be performed in parallel. The first ten steps all contain a Montgomery modular multiplication that is sometimes accompanied by a modular addition, while the 11th step only consists of a modular addition. This makes the total execution time $10T_{MMM} + 1$, with T_{MMM} the latency of one Montgomery modular multiplication. In the same way, it can be found that the total execution time of one EC point addition is $14T_{MMM} + 1$.

Algorithm 3 EC Point Doubling and Addition

Require: $P_1 = (X_1, Y_1, Z_1)$
Ensure: $2P_1 = (X_3, Y_3, Z_3)$

1. $T_1 \leftarrow Z_1^2$
2. $T_2 \leftarrow X_1^2$
3. $Z_3 \leftarrow T_1 T_2$
4. $T_1 \leftarrow T_1^2$
5. $T_2 \leftarrow T_2^2$
6. $T_1 \leftarrow bT_1$
7. $T_2 \leftarrow Y_1^2$ $\quad X_3 \leftarrow T_1 + T_2$
8. $T_3 \leftarrow aZ_3$ $\quad T_2 \leftarrow T_2 + T_1$
9. $T_1 \leftarrow T_1 Z_3$ $\quad T_3 \leftarrow T_3 + T_2$
10. $T_3 \leftarrow X_3 T_3$
11. $Y_3 \leftarrow T_3 + T_1$

(a)

Require: $P_1 = (x, y, 1)$, $P_2 = (X_2, Y_2, Z_2)$
Ensure: $P_1 + P_2 = (X_3, Y_3, Z_3)$

1. $T_1 \leftarrow Z_1^2$
2. $T_2 \leftarrow Y_2 T_1$
3. $T_3 \leftarrow X_2 Z_1$ $\quad T_2 \leftarrow T_2 + Y_1$
4. $T_1 \leftarrow aT_1$ $\quad T_3 \leftarrow T_3 + X_1$
5. $T_4 \leftarrow Z_1 T_3$
6. $T_3 \leftarrow T_3^2$ $\quad T_1 \leftarrow T_4 + T_1$
7. $T_1 \leftarrow T_3 T_1$
8. $Z_3 \leftarrow T_4^2$
9. $T_4 \leftarrow T_2 T_4$
10. $T_2 \leftarrow T_2^2$ $\quad T_1 \leftarrow T_1 + T_4$
11. $T_1 \leftarrow X_2 Z_3$ $\quad X_3 \leftarrow T_2 + T_1$
12. $T_2 \leftarrow Y_2 Z_3$ $\quad T_1 \leftarrow X_3 + T_1$
13. $T_1 \leftarrow T_4 T_1$ $\quad T_2 \leftarrow X_3 + T_2$
14. $T_3 \leftarrow Z_3 T_2$
15. $\quad Y_3 \leftarrow T_1 + T_3$

(b)

4.3 Modular multiplicative inverter

Modular multiplicative inversion is computed using Fermat's theorem [13, 9], $a(x)^{-1} = a(x)^{2^m-2} \bmod p(x)$. This modular exponentiation of $a(x)$ by $2^m - 2 = (1, 1, \cdots 1, 1, 0)_2$, using the square-and-multiply algorithm [9], is shown in Algorithm 4.

Algorithm 4 Modular multiplicative inversion

Require: polynomial $a(x)$, $0 \leq a(x) < p(x)$ and $p(x)$
Ensure: $b(x) = a(x)^{2^m-2} \bmod p(x) = a(x)^{-1} \bmod p(x)$

1: $b(x) \leftarrow a(x)$
2: **for** i from 2 to $m-2$ **do**
3: $\quad b(x) \leftarrow b(x)b(x) \bmod p(x)$
4: $\quad b(x) \leftarrow b(x)a(x) \bmod p(x)$
5: **end for**
6: $b(x) \leftarrow b(x)b(x) \bmod p(x)$

The MMI circuit controls the execution of the square-and-multiply algorithm. It gets its START signal from the PtoA circuit. The MMI circuit then commands the MMMC to perform a MMM twice in every loop iteration in Algorithm 4 and once at the end.

4.4 Affine to projective representation converter

As explained in Section 3.2, it is more efficient to do point operations using projective instead of affine coordinates. The conversion to projective coordinates is as follows:

$$(xp(x),yp(x)) \rightarrow (X(x),Y(x),Z(x)) = (xp(x),yp(x),1) \quad (3)$$

where $(xp(x),yp(x))$ is the point in affine coordinates and $(X(x),Y(x),Z(x))$ is the same point in projective coordinates.

4.5 Normal to Montgomery representation converter

The conversion of $a(x)$ from the normal to the Montgomery representation is computed as $Mont(a(x), r(x)^2) = a(x)r(x) \bmod p(x)$. Multiplication by the MMMC of two polynomials that are in Montgomery representation will produce the Montgomery representation of the product as $Mont(a(x)r(x), b(x)r(x)) = a(x)b(x)r(x) \bmod p(x)$.

Modular addition of two polynomials that are in Montgomery representation will produce the Montgomery representation of the sum as $a(x)r(x) \bmod p(x) + b(x)r(x) \bmod p(x) = (a(x) + b(x))\, r(x) \bmod p(x)$. Because of these relations, the Montgomery representation of the coordinates of P, the coefficients a, b and the number 1 will be calculated in the beginning of the point multiplication by the NtoM circuit and all the operations during the EC point multiplication will be performed in Montgomery representation.

The conversion to Montgomery representation of the number one is computed as $Mont(1, r(x)^2) = r(x) \bmod p(x)$.

The other conversions in NtoM are done by the following four operations:

$$\begin{aligned} Mont(a(x), r(x)^2) &= a(x)r(x) \bmod p(x) \\ Mont(b(x), r(x)^2) &= b(x)r(x) \bmod p(x) \\ Mont(xp(x), r(x)^2) &= xp(x)r(x) \bmod p(x) \\ Mont(yp(x), r(x)^2) &= yp(x)r(x) \bmod p(x)\,. \end{aligned}$$

4.6 EC point multiplier

The EPM circuit controls the execution of Algorithm 1. In every iteration of the loop, an EPD is executed. An EPA is only performed when the evaluated key bit is 1.

4.7 Projective to affine coordinates converter

After completing the EC point multiplication the result point Q must be converted from projective coordinates to affine coordinates. This is done as $(X(x), Y(x), Z(x)) \rightarrow (xq(x),\ yq(x))$ such that $xq(x) = X(x)Z(x)^{-1}$ and $yq(x) = Y(x)\ Z(x)^{-2}$ [8]. PtoA controls four operations in the following order:

$$\begin{aligned} Z(x)^{-1}r(x) \bmod p(x) &= MMI\ of\ Z(x) \\ Mont(X(x)r(x), Z(x)^{-1}r(x)) &= xq(x)r(x) \bmod p(x) \\ Mont(Z(x)^{-1}r(x), Z(x)^{-1}r(x)) &= Z(x)^{-2}r(x) \bmod p(x) \\ Mont(Y(x)r(x), Z(x)^{-2}r(x)) &= yq(x)r(x) \bmod p(x)\,. \end{aligned}$$

4.8 Montgomery to normal representation converter

Because the coordinates of the product point must be in normal representation, as a last action a conversion from Montgomery representation to normal representation is needed. This conversion requires two additional executions of the MMMC operation with the inputs $xq(x)r(x) \bmod p(x)$ and 1, then $yq(x)r(x) \bmod p(x)$ and 1, as $xq(x) = Mont(xq(x)\ r(x),\ 1)$, $yq(x) = Mont(yq(x)r(x), 1)$.

5. IMPLEMENTATION RESULTS

The EC processor has been implemented on a Xilinx Virtex XCV800-4-HQ240 FPGA. The implementation results are given in Table 1. The number of gates in the table are

Table 1: Area and latency results of 160-bit implementations (for $hw = 80$)

w	1	4	8
Number of gates	138 528	149 037	150 678
Clock Period (ns)	19.956	19.977	20.923
Latency of ECP (ms)	9.652	7.249	3.801

not the same as ASIC gates. They are equivalent gates computed by Xilinx Project Manager. The table shows that the minimum clock period and the area of the circuit decrease when the word length w is enlarged. When using a larger w, the latency of the circuit becomes lower, because the point multiplication can be done in less clock cycles.

The maximum clock frequency is independent of the bit length m. The total latency of the ECP can be calculated as $(1900 + 14hw)\,T_{MMMC} + 316 + 2hw$, where hw is the Hamming Weight of the key and T_{MMMC} is the latency of one MMM. $T_{MMMC} = m$ for $w = 1$ and $T_{MMMC} = 3\frac{m}{w}$ for $w > 1$. One elliptic curve point multiplication (ECPM) requires 3.810 ms at 47 MHz. Because there is no previous work which uses modular Montgomery multiplication method and all of the used platforms are different from the platform we use, it is hard to compare the area of the current work with the previous ones. The time needed for one ECPM with our design is approximately the same as the result reported in [4], smaller than the results reported in [14, 15, 16, 17, 18, 19, 20]. Unfortunately our circuit is slower than the ones reported in [21] and [22].

6. CONCLUSIONS

We have described an efficient implementation of an elliptic curve processor over $GF(2^m)$. The processor can be programmed to execute a modular multiplication, addition, multiplicative inversion, EC point addition/doubling and multiplication. Our architecture uses the Montgomery method for modular multiplication because of its implementation advantages. It allows to use a systolic array which makes the clock frequency independent of the bit length m.

7. REFERENCES

[1] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.

[2] N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[3] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[4] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An implementation of elliptic curve cryptosystem over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.

[5] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle. Hardware architectures for public key cryptography. *Elsevier Science Integration the VLSI Journal*, 34(1-2):1–64, 2003.

[6] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[7] I. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[8] J. López and R. Dahab. Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. Technical Report IC-98-39, Institute of Computing, State University of Campinas, Campinas, 13081-970 Sao Paulo, Brazil, October 1998.

[9] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[10] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44:519–521, 1985.

[11] Ç. K. Koç and T. Acar. Montgomery multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14:57–69, 1998.

[12] N. Mentens, S. B. Örs, B. Preneel, and J. Vandewalle. An FPGA implementation of a Montgomery multiplier over GF(2^m). 2004. DDECS, Stara Lesna, Slovakia, 18-21 April, 8 pages, to appear.

[13] N. Koblitz. *A Course in Number Theory and Cryptography*, volume 114 of *Graduate text in mathematics*. Springer-Verlag, Berlin, Germany, second edition, 1994.

[14] S. Sutikno, R. Effendi, and A. Surya. Design and implementation of arithmetic processor GF(2^{155}) for elliptic curve cryptosystems. In *Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'98)*, pages 647–650, 1998.

[15] L. Gao, S. Shrivastava, H. Lee, and G. E. Sobelman. A compact fast variable key size elliptic curve cryptosystem coprocessor. In *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 304–305, 1999.

[16] L. Gao, S. Shrivastava, and G. E. Sobelman. Elliptic curve scalar multiplier design using FPGAs. In Ç. K. Koç and C. Paar, editors, *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 1717 in Lecture Notes in Computer Science, pages 257–305, Worcester, MA, USA, August 1999. Springer-Verlag.

[17] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong. FPGA implementation of a microcoded elliptic curve cryptographic processor. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM)*, pages 68–76, 2000.

[18] M. Ernst, S. Klupsch, O. Hauck, and S. A. Huss. Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems. In *Proceedings of 12th IEEE Workshop on Rapid System Prototyping*, pages 24–31, Monterey, CA, June 2001.

[19] J. Goodman and A. P. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808–1820, November 2001.

[20] S. Janssens, J. Thomas, W. Borremans, P. Gijsels, I. Verbauwhede, F. Vercauteren, B. Preneel, and J. Vandewalle. Hardware/software co-design of an elliptic curve public-key cryptosystem. In *Proceedings IEEE Workshop on of Signal Processing Systems*, pages 209–216, 2001.

[21] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for GF(2^m). In Ç. K. Koç and C. Paar, editors, *Proceedings of 2nd International Workshop on Cryptograpic Hardware and Embedded Systems (CHES)*, number 1965 in Lecture Notes in Computer Science, pages 41–56, Worcester, Massachusetts, USA, August 17-18 2000. Springer-Verlag.

[22] H. Eberle, N. Gura, and S. Chang-Shantz. A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$. In *ASAP 2003*, The Hague, The Netherlands, June 2003 2003.