GF(2m) Arithmetic Modules for Elliptic Curve Cryptography

Conference Paper · October 2006

DOI: 10.1109/RECONF.2006.307768 · Source: IEEE Xplore

2 authors:

Miguel Morales-Sandoval
Center for Research and Advanced Studies of the National Polytechnic Institute
79 PUBLICATIONS **229** CITATIONS
SEE PROFILE

Claudia Feregrino
Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
136 PUBLICATIONS **532** CITATIONS
SEE PROFILE

$GF(2^m)$ Arithmetic Modules for Elliptic Curve Cryptography

Miguel Morales-Sandoval and Claudia Feregrino-Uribe
National Institute for Astrophysics, Optics and Electronics
Computer Science Department
Luis Enrique Erro No. 1, Sta. Ma. Tonantzintla, 72840 Puebla, México
{mmorales, cferegrino}@inaoep.mx

Abstract

This paper reports work in progress in the design, implementation and evaluation of a reconfigurable finite field arithmetic architecture with a direct application in Elliptic Curve Cryptography (ECC) for mobile devices. This module contributes to manage the current interoperability problems in ECC, that are due to the several choices in the implementation of ECC cryptosystems. We report an evaluation of some finite field arithmetic modules in an architecture for computing scalar multiplication, which is the most time consuming in ECC cryptographic schemes. The arithmetic modules were evaluated for all the $GF(2^m)$ NIST elliptic curves in a hardware architecture implemented in field programmable technology.

1. Introduction

The tendency in communications is the use of mobile devices in a global communication world and the pervasive and ubiquitous computing. These same devices are forming Ad-Hoc networks and interacting with other devices to perform an specific task. Examples of this kind of networks are sensor, automotive and personal networks. In this heterogeneous inter-networked device environment, *security* and *interoperability* are two key aspects to take into account.

Classical information security services like confidentiality, authentication, integrity and no-repudiation are not straightly implemented in the wireless networks due to the complexity of the cryptographic algorithms and the constrained computational resources in the mobile devices. Authentication and confidentiality are two of the most required security services in wireless communication for the following scenarios:

- Current wireless networks use radio signals to communicate which makes any device can pick-up unprotected signals and capture information. Confidentiality is required to ensure the privacy of information exchange.
- Mobile devices need to authenticate every time they connect to a network or to a service provider in order to guarantee that they are who they claim to be.

Elliptic Curve Cryptography (ECC) [10], [15] is a kind of public key cryptosystem that guarantees all the above security services using shorter keys while provides the same security level than other widely used cryptosystems, for example RSA [17]. The use of shorter length keys implies less space for key storage, time saving when keys are transmitted and less costly arithmetic computations. These characteristics make ECC the best and default choice to provide security in wireless networks [11].

But ECC is more complex. Instead of a single encryption algorithm like RSA, ECC can be implemented in different ways. At the same time, the related arithmetic is more complex. Modular and elliptic curve arithmetic is performed using operands of the same size in bits than the key being used. Currently, a secure key size in ECC is greater than 163 bits. Arithmetic algorithms are the core operations of high level security algorithms like encryption (for confidentiality) or digital signatures (for authentication). The interoperability problem [23] related to ECC is due to the selection of an underlaying finite field, a security level, the field representation, arithmetic algorithms, an elliptic curve and the coordinate system. Such problem must be addressed in order to have a global security solution based on ECC technology.

Although there have been reported security hardware and software solutions based on ECC, only a few of them intent to solve the interoperability problem. While some of the hardware solutions require more hardware area which makes them unsuitable for constrained devices, the software ones are implemented on workstations where the computational resources are not constrained as in the wireless devices. A careful study of the best choices to meet throughput and area requirements for mobile applications is un-

1-4244-0690-0/06/$20.00 ©2006 IEEE.

clear, making it difficult to provide an efficient and interoperable ECC security solution for mobile communications.

This paper reports work in progress toward the design of a reconfigurable ECC security architecture that adapts to changes in the ECC parameters while achieves the higher throughput with minimal power consumption. We report an evaluation of some of the finite field arithmetic algorithms in a generic architecture for computing the most time consuming operation in ECC, the scalar multiplication.

2. Related work

The main problem for interoperability in elliptic curve cryptography is the variety of ECC parameters that can be used for implementation. An ECC cryptosystem is defined as the tuple $T = (GF(q), a, b, G, n, h)$, where $GF(q)$ is a finite field, a and b define the elliptic curve on $GF(q)$, G is a generator point of the elliptic curve, n is the order of G, that is, the smaller integer such that $nG = O$ (identity point in the additive group). h is called the co-factor and it is equal to the total number of points in the curve divided by n.

Security services are provided by ECC cryptographic schemes for key agreement, digital signatures and bulk encryption. The most time consuming operation is such schemes is the scalar multiplication, which is intrinsically related to the tuple T. This costly elliptic curve operation is performed according three layers:

At the top layer there are different methods for computing the scalar multiplication independently of the selected finite field. An scalar multiplication is the result of adding the point P to itself $n-1$ times. That is,

$$kP = \underbrace{P + P + P + \cdots + P}_{k\ times}$$

This operation is a consecutive sum of points that can be performed using two kinds of sums: the ECC-ADD, which consist of the sum of two different points $(P + Q)$ and ECC-DOUBLE, which consist of the sum of the same point $(P + P)$.

In the middle layer is the coordinate system being used. Depending on this representation the ECC-ADD and ECC-DOUBLE are defined in a different way. The simplest is the affine representation (x, y) but most of the reported ECC implementations use projective coordinates because of point addition is free of finite field inversions, being this operation the most time consuming.

At the lower layer is the finite field arithmetic. The performance of the arithmetic units impacts the overall performance of the scalar multiplication and hence the performance of the ECC cryptographic schemes. Finite field operations are multiplication, inversion, squaring and adding. Depending on the finite field used, these operation are performed in different ways. Also, several algorithms to perform these operations are reported but only a few of them have been evaluated in order to discriminate them for specific applications.

While the number of ECC-ADD and ECC-DOUBLE operations depends on the method chosen in the top layer, the kind and number of operations in the finite field depends on the coordinates used in the middle layer.

Efficient hardware/software implementations of the scalar multiplication kP have been the main research topic on ECC in recent years. Tables 1 and 2 summarizes the algorithms used in each layer of kP and the timing achieved. Table 3 shows the different approaches to implement the three layers of kP computation.

The main focus in the related works has been the performance of the kP scalar multiplication. Our approach is different. We aim to perform this operation as fast as possible but keeping a flexible architecture that can adapts to several security levels. This implies a careful algorithms selection and implementation, that leads to a dedicated unit for scalar multiplication that performs well for several elliptic curves and finite fields. Table 1 shows how different algorithm selection in each kP layer will lead to different area/timing results. It is noted that several multipliers have been used at different levels of parallelism. It is not clear how these choices will impact the area resources of the coprocessor and what are the advantages of using one of the reported multipliers: Karatsuba, LFRS, Massey Omura (M-O), digit-serial (D-S). The same applies for the inversion and square algorithms. This table also shows how current ECC hardware solutions are not interoperable among them (different elliptic curves). Although in some works the design of the arithmetic units is parameterizable in the order field, the architecture needs to be reconfigured out of line for other finite fields orders. It would be desired in mobile devices a real time adaptation of the architecture to different security levels. We aim to provide such architecture by using dynamic reconfiguration.

2.1. Finite field arithmetic

Due to its mathematical properties, binary $GF(2^m)$ and prime $GF(p)$ fields have been widely used in cryptography. For $GF(2^m)$, finite field operations depend on a basis, which can be polynomial, normal or dual. In polynomial basis, the elements of $GF(2^m)$ are viewed as $m-1$ grade polynomials $A(x)$ with coefficients in $GF(2) = \{0, 1\}$. A basis of $GF(2^m)$ is one of the form $\{1, t, t_1, t_2, t_{m-1}\}$, where t is a square of an irreducible m grade polynomial $P(x)$ (cannot be factored as two polynomials). Arithmetic in $GF(2^m)$ with polynomial basis is arithmetic of polynomials modulo $P(x)$. In normal basis, the irreducible polynomial is not used. There is always a trade-off when using different bases for both software and hardware implementations. Squaring

Ref.	m	kP **method**	**Coordinates**	**Basis**	**Multiplier**	**Timing**
Co-processors						
	270				M-O (3)	6.8
[5]	191	D&A	Projective	ONB	M-O (5)	2.3
	155				M-O (7)	1.2
[1]	191	D&A	Jacobian	Polynomial	LFSR (4)	3.7
[4]	113	2P	Projective	Polynomial	Karatsuba	10.9
	151					5.1
[9]	176	D&A	Affine	Polynomial	ABC proc.	6.9
	191					8.2
	239					12.8
		D&A				0.26
[13]	163	D&A NAF	López-Dahab	Polynomial	D-S(41)	0.23
		D&A τ-adic NAF				0.07
[19]	191	Montgomery	López-Dahab	Polynomial	Karatsuba Ofman	0.05
[2]	113	Montgomery	López-Dahab	ONB	ONB Bit-serial (2)	0.27
	113					3.7
[12]	155	D&A	Affine	ONB	ONB Multiplier	6.8
	281					14.4
Processors						
					(4) D-S	0.96
		D&A	Jacobians		(8) D-S	0.61
[16]	167			Polynomial	(16) D-S	0.36
					(4) D-S	0.55
		Montgomery	López-Dahab		(8) D-S	0.35
					(16) D-S	0.21
	K163					0.14
	k193					0.18
[6]	k233	Montgomery	López-Dahab	Polynomial	D-S	0.22
	163					1.5
	193					1.83
	233					2.21
[20]	160	Addition-Sub chain NAF	Jacobians	Polynomial	64-*bit* dual field Montgomery multiplier	0.19
[14]	160	D&A	López-Dahab	Polynomial	Systolic Montgomery multiplier	3.810
[8]	-	New 3 bits at a time	Projective	Polynomial	NA	
Software						
	163					2.3
	233				Karatsuba	4.7
[22]	283	Windowed NAF	Proyective	Polynomial		9.5
	409				Window Comb	19.8
	571				Window Comb	44.9
	163					1.68
[7]	233	Fix Base Comb	Proyective	Polynomial	-	3.96
	283					5.91

Table 1. Approaches taken in ECC hardware and software implementations in GF(2^m)

is easier in normal basis but inversion is slower than in polynomial basis [2]. Orlando and Paar [16] states that multipliers that use normal basis are prohibitively expensive in terms of area when that order is high (m = 400, 500). Optimal normal basis shows some improvements in efficiency but there are few fields for which there exist this kind of basis. Hankerson [7] and López and Dahab [3] reports a survey of methods (the above described) to compute the finite field arithmetic for both $GF(2^m)$ and $GF(p)$.

3. Architecture for scalar multiplication

3.1. The kP top layer

The operation kP, being k an element of the finite field $GF(q)$ and P a point in the elliptic curve defined on $GF(q)$

Ref.	m	Device	Area	Freq	Time $(msec)$
			Co-processors		
	155		63%	37 MHz	1.2
[5]	191	XC4085XLA	69%	36 MHz	2.3
	270		82%	34 MHz	6.8
[1]	191	XCV1000	NA	50 MHz	3.72
[4]	113	AT40K Amtel (2304 CLB, 40 Kgates)	96%	12 MHz	10.9
[9]	¡255	XCV2000E	20%(4048 Slices, 74103 gates)	40MHz	5.1
[13]	K163	XCV2000E	(D = 41)10,017 LUTs 1930 FF	66 MHz	0.075
[19]	191	VirtexE 3200	56.39% (18314 CLB slices)	9.9 MHz	0.054
[2]	113	XC2V6000	6961 Slices	56 MHz	0.27
	113		1290 Slices	45 MHz	3.7
[12]	155	XCV300-4	1567 Slices	36 MHz	6.8
	281		2622 Slices	33 MHz	14.4
			Processors		
[16]	167	XCV400E-8	D = 4-16; 1627-3002 LUTs	85.7-76.7 MHz	0.55-0.36
[6]	< 255	XCV2000E	-	66.4 MHz	-
[20]	160	ASIC	118 Kgates	510 MHz	0.19
[14]	160	XCV800	138 - 150 Kgates	47 MHz	3.81

Table 2. Devices used, area consumption and execution time in ECC implementations in $GF(2^m)$

is the most time and computational demanding operation in ECC cryptographic schemes. All the proposed methods for computing this operation use the binary representation of k and for each bit value they perform one of two addition rules: ECC-ADD and ECC-Double. Addition of points has a geometrical interpretation and its definition varies depending on the coordinates being used to represent the elliptic points.

The simplest way to compute kP is applying the binary or standard method, which is defined as follows:

```
**Binary method for scalar multiplication** $kP$
**Input:** $P = (x, y)\ x, y \in GF2^m$ and $k = (k_{m-1}, k_{m-2}, ..., k_0)$
**Output:** $R = kP$
$R \leftarrow (0, 0)$
$S \leftarrow P$
**for** $i$ **from** 0 **to** $m - 1$
    **if** $k_i = 1$
        $R \leftarrow$ ECC-ADD$(R, S)$
    **end if**
    $S \leftarrow$ ECC-Double$(S)$
**end for**
```

3.2. The kP middle layer

Using affine representation of elliptic curve points, the ECC-ADD operation of two points $P = (x_1, y_1)$, $Q = (x_2, y_2)$ is the point $R = (x_3, y_3)$ where:

$x_3 \leftarrow \lambda^2 + \lambda + x_1 + x_2 + a$
$y_3 \leftarrow \lambda(x_1 + x_3) + x_3 + y_1$
$\lambda \leftarrow (y_2 + y_1)/(x_2 + x_1)$

The ECC-DOUBLE operation is $R = (x_3, y_3) = P + P = 2P$ where:

$x_3 \leftarrow \lambda^2 + \lambda + a$
$y_3 \leftarrow x_1^2 + \lambda x_3 + x_3$
$\lambda \leftarrow x_1 + y_1/x_1$

So, for ECC-ADD it is necessary to perform two field multiplications, one squaring and one inversion. The ECC-Double requires one extra squaring operation. It es well known that projective coordinates avoids the inversion operation in each ECC point addition but introduces more field multiplication (seven for ECC-ADD and twelve for ECC-Double). It can be beneficial from the point of view of timing but the inversion is required because only one inversion operation is necessary at the end of the kP computation. So, in a hardware implementation, the inversion module is not avoided despite projective coordinates be used.

From this selection at the first and second layer of kP computation, we propose a generic architecture that consists of two dedicated units for ECC-ADD and ECC-Double commanded by a control unit that performs the binary algorithm for kP. Such architecture is shown in figure 1.

The multiplier, division and squaring modules can be updated just by replacing them by better performed modules. We select a field serial multiplier and compared with a digit serial multiplier, we implemented the combinatorial squarer proposed in [13] and design an architecture for the finite field direct division algorithm proposed in [21].

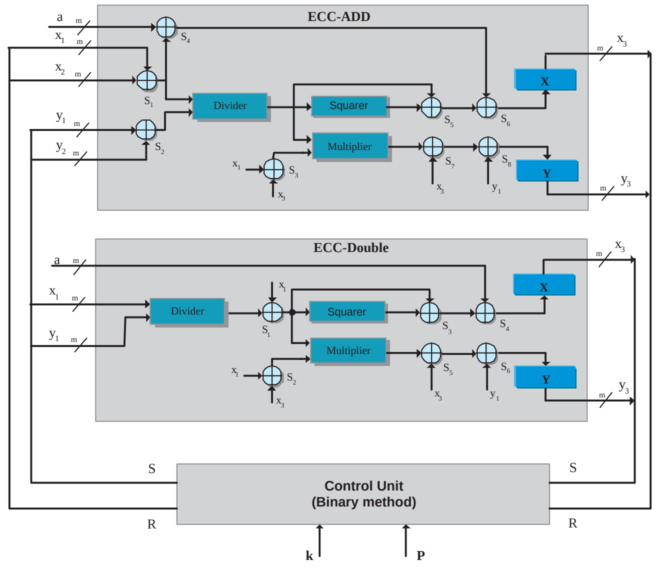


Figure 1. Architecture for ECC-ADD and ECC-DOUBLE

3.3. The kP lower layer

We work with the binary field $GF(2^m)$ using polynomial basis because of according to the literature, it leads to efficient hardware implementations of ECC. Due that we are targeting our hardware architecture to mobile devices, we are interested in algorithms that consumes few resources and hence, power consumption. We selected the the field serial multiplier algorithm that computes $a(x)b(x) \bmod F(x)$ in m iterations. Then, we added parallelism to the multiplier and we compare the gain in performance at the cost of area.

```
$GF(2^m)$ **serial multiplication algorithm**
**Input:** $A(x), B(x) \in GF2^m$, $F(x)$ the irreducible polynomial of degree $m$
**Output:** $C(x) = A(x)B(x) \bmod F(x)$
$C(x) \leftarrow 0$
**for** $i$ **from** $m-1$ **down to** $0$
    $C(x) \leftarrow C(x)x + A(x)b_i + c_{m-1}P(x)$
**end for**
```

This algorithm can be improved gradually if instead of considering just one bit of the polynomial $B(x)$, a group of bits is considered at each step of the field multiplication algorithm. The number of bits considered is called the digit d. So, a field multiplication computed by a digit serial algorithm is performed as follows:

```
$GF(2^m)$ **digit-serial multiplication algorithm**
**Input:** $A(x), B(x), F(x) \in GF2^m$, $d$ the digit size and $s = \lceil m/d \rceil$
**Output:** $C(x) \leftarrow A(x)B(x) \bmod F(x)$
$C(x) \leftarrow B_{s-1}(x)A(x) \bmod F(x)$
**for** $i$ **from** $s-2$ **down to** $0$
    $C(x) \leftarrow x^d C(x)$
    $C(x) \leftarrow B_k(x)A(x) + C(x) \bmod F(x)$
**end for**
```

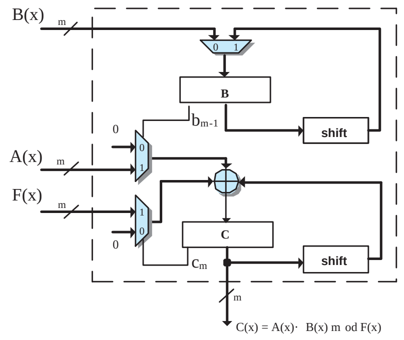
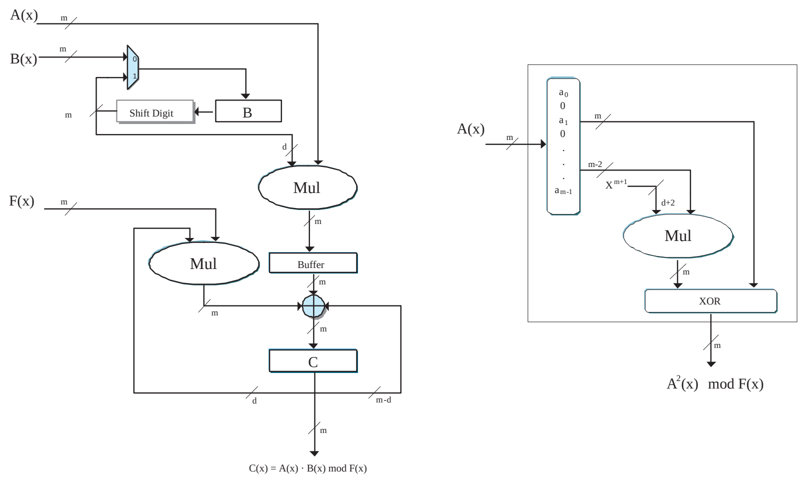


Figure 2. Hardware architecture for serial finite field multiplication

The digit-serial algorithm reduces the field multiplication computation to in m/d iterations but introduces complexity in each step of the multiplication. For these two kinds of multipliers, we designed two architectures shown in figure 2 and 3. The squaring module is taken from [13] which computes $A(x)^2$ in just one clock cycle.

For inversion, we considering the Modified Almost Inverse algorithm, the Fermant's Theorem and the algorithm for direct division proposed in [21]. The Fermant's theorem reuses the multiplier and the squaring modules but requires $m-1$ iterations. This results in $(m-1)$xm iterations if the serial multiplier is used or $(m-1)$x(m/d) iterations if



Finite field digit-serial multiplier — Finite field combinatorial squarer

Figure 3. Hardware architecture for digit serial finite field multiplication

the digit serial multiplier is used. On the contrary, the Modified Almost Inverse algorithm has a complexity similar to the direct division algorithm which instead of computing the inverse of a field element $A(x)$, it performs the operation $A(x)/B(x)$. Both methods perform at most $2m-1$ iterations but in the first one it is necessary to perform a multiplication that requires extra clock cycles. We implemented the direct division algorithm defined as follows:

```
$GF(2^m)$ division algorithm
Input: $X_1(x), Y_1(x) \in F_{2^m}$, $X_1(x) \neq 0$ and $F(x)$ the
irreducible polynomial of degree $m$
Output: $U(x) = Y_1(x)/X_1(x)$ mod $P(x)$
$A(x) \leftarrow X_1(x)$
$B(x) \leftarrow F(x)$
$U(x) \leftarrow Y_1(x)$
$V(x) \leftarrow 0$
while $A(x) \neq B(x)$ do
    if $x$ divides to $A(x)$
      $A(x) \leftarrow A(x)x^{-1}$
      if $x$ divides to $U(x)$ then $U(x) \leftarrow U(x)x^{-1}$
      else $U(x) \leftarrow (U(x)+F(x))x^{-1}$
      end if
    else if $x$ divides to $B(x)$ then
        $B(x) \leftarrow B(x)x^{-1}$
        if $x$ divides to $V(x)$ then $V(x) \leftarrow V(x)x^{-1}$
        else $V(x) \leftarrow (V(x)+F(x))x^{-1}$
        end if
    else if $deg(A(x)) > deg(B(x))$ then
        $A(x) \leftarrow (A(x)+B(x))x^{-1}$
        $U(x) \leftarrow U(x)+V(x)$
        if $x$ divides to $U(x)$ then $U(x) \leftarrow U(x)x^{-1}$
        else $U(x) \leftarrow (U(x)+F(x))x^{-1}$
        end if
    else
        $B(x) \leftarrow (A(x)+B(x))x^{-1}$
        $V(x) \leftarrow U(x)+V(x)$
        if $x$ divides to $V(x)$ then $V(x) \leftarrow V(x)x^{-1}$
        else $V(x) \leftarrow (V(x)+F(x))x^{-1}$
        end if
    end if
end while
```

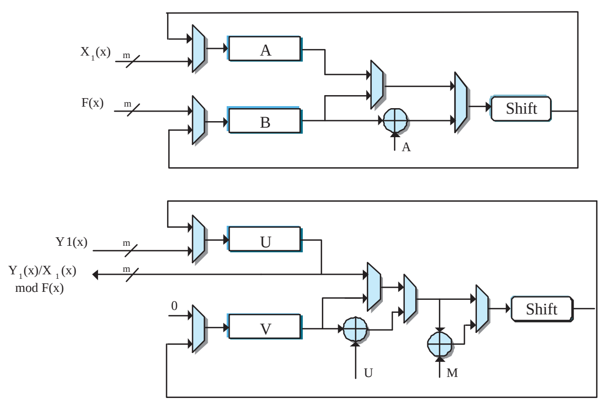


Figure 4. Architecture for division in F_{2^m}

The architecture for the above division algorithm is depicted in figure 4.

4. Results

We evaluated all the above algorithms in our generic kP architecture. All the $GF(2^m)$ arithmetic modules were described in VHDL and are fully parameterizable in the field order m. We design input and output interface modules to allow a 32-bit interface with a host processor. All the arithmetic modules for $GF(2^m)$ were synthesized for a Xilinx XC2V2000 FPGA. The kP architecture was synthesized for the binary field defined by $m \in \{163, 233, 283\}$. For the case of $m = 233, 283$ we need to use the XC2V4000 FPGA due to higher area requirements.

Figures 5 shows the area resources requirements for the multiplier, using the serial and digit version. The division and squaring area consumption is shown in figure 6. It can be seen the linear in area resources with respect to the security level used.

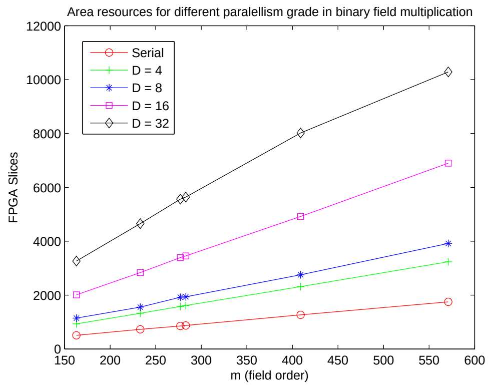


Figure 5. Area resources for $GF(2^m)$ multiplication for different m values

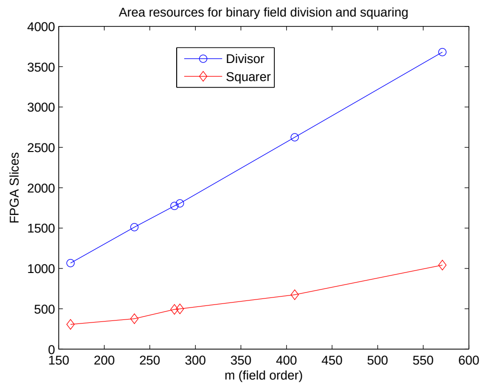


Figure 6. Area resources for $GF(2^m)$ multiplication and squaring for different m values

Using the arithmetic modules, we evaluate them for the generic elliptic curve for $m = 163$ recommended in SEC-1. Figure 7 shows the timing spent for kP using the five different field multiplication modules. As we can see, different to the assumption that a greater digit will perform the kP operation faster, we found that the best timing for kP is achieved when using a digit size of 8. This is due to the higher complexity in the multiplier which produce a larger clock cycle period. This observation is supported by the values listed in table 3.

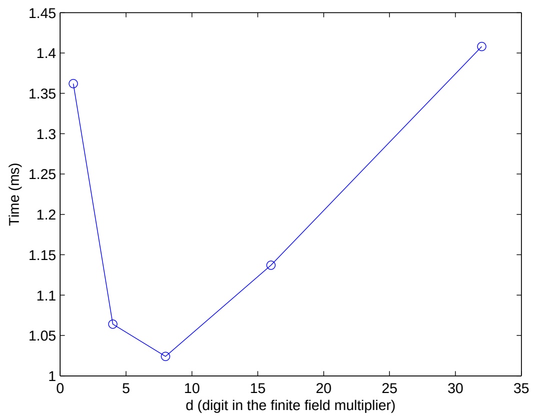


Figure 7. Time (ms) for scalar multiplication for $m = 163$ using different field multipliers

We validated all the architectures comparing results against a software implementation that is a little modification of the code available in [18].

5. Concluding remarks

We presented $GF(2^m)$ arithmetic modules for a reconfigurable Elliptic Curve Cryptography architecture. We evaluated a serial and digit serial multiplier, a division algorithm and a combinatorial squaring unit. We reported the area requirements and the timing performance achieved. We are going to use other arithmetic module for comparison, among them, the Fermat Algorithm and the Ito-Tsuji algorithms for inversion. This is in order to have a better way to select the ones arithmetic modules that will be considered in the final reconfigurable architecture.

Acknowledgments

First author thanks the National Council for Science and Technology (CONACyT) for financial support throw the scholarship number 171577.

Multiplier	Area	Cycles/kP	Clk period (*ns*)	Time kP(*ms*)
Serial	5632 (52%)	72527	19.409	1.36
D = 4	6762 (62%)	52620	21.611	1.06
D = 8	7342 (68%)	49360	20.747	1.02
D = 16	8537 (79%)	47730	22.286	1.13
D = 32	10750 (99%)	46915	29.040	1.4

Table 3. Area/timing results for kP using different field multipliers

References

[1] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. Reconfigurable implementation of elliptic curve crypto algorithms. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, pages 284–291, Washington, DC, USA, 2002. IEEE Computer Society.

[2] R. Cheung, N. Telle, W. Luk, and P. Cheung. Customizable elliptic curve cryptosystems. *IEEE Trans. on VLSI Systems*, 13(9):1048–1059, Sept. 2005.

[3] R. Dahab and J. López. An Overview of Elliptic Curve Cryptography. Technical Report IC-00-10, State University of Campinas, Brazil, May 2000.

[4] M. Ernest *et al.* A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^n). In *Proc. of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 381–399, Redwood Shores, CA, August 2002. Springer.

[5] M. Ernest *et al.* Rapid Prototyping for Hardware Accelerated Elliptic Curve Public Key Cryptosystems. In *Proc. of 12th IEEE Workshop on Rapid System Prototyping, RSP'2001*, pages 24–31, Monterey, CA, June 2001.

[6] N. Gura *et al.* An End to End Systems Approach to Elliptic Curve Cryptography. In *Proc. of CHES 2002*, volume 2523, pages 349–365. Springer, 2002.

[7] D. Hankerson, L. López, and A. Menezes. Software Implementation of Elliptic Curve Cryptography over Binary Fields. In *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24, Worcester, MA, August 2000. Springer.

[8] A. Hodjat, D. D. Hwang, and I. Verbauwhede. A scalable and high performance elliptic curve processor with resistance to timing attacks. In *ITCC'05: International Conference on Information Technology: Coding and Computing*, volume I, pages 538–543, 2005.

[9] T. Kerins *et al.* Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m). In *Proc. of 12th International Conference on Field Programmable Logic and Application, FPL'2002*, volume 2438 of *Lecture Notes in Computer Science*, pages 750–759, Montpellier, France, September 2002. Springer.

[10] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, November 1987.

[11] K. Lauter. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless Communications*, pages 62–67, 2004.

[12] P. Leong and K. H. Leung. A Microcoded Elliptic Curve Processor Using FPGA Technology. *IEEE Trans. on VLSI Systems*, 10(5):550–559, October 2002.

[13] J. Lutz and A. Hasan. High performance fpga based elliptic curve cryptographic co-processor. In *ITCC'04: International Conference on Information Technology: Coding and Computing*, volume 2, page 486, 2004.

[14] N. Mentens, S. Berna, and B. Preneel. An FPGA Implementation of an Elliptic Curve Processor *gf*(2^m). In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 454–457, Boston, MA, 2004.

[15] V. Miller. Use of Elliptic Curves in Cryptography. In *Proc. of Advances in Cryptology, CRYPTO'85*, pages 417–426, Santa Barbara, CA, August 1985.

[16] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 41–56, Worcester, MA, August 2000. Springer.

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

[18] M. Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications, 1999.

[19] N. Saquib, F. Rodriguez, and A. Diaz. A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication over GF(2^n). In *Proc. of 11th Reconfigurable Architectures Workshop, RAW'04*, pages 26–27, Sta. Fe, USA, April 2004.

[20] A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *Transactions on Computers*, 52(4):449–460, April 2003.

[21] Shantz, S. C. From Euclid's GCD to Montgomery Multiplication to the Great Divide. Technical Report TR-2001-95, Sun Microsystems Laboratories, 2001.

[22] A. Weimerskirch, D. Stebila, and S. C. Shantz. Generic $gf(2^m)$ arithmetic in software and its application to ecc. In *The Eighth Australasian Conference on Information Security and Privacy (ACISP 2003)*, volume 2727 of *Lecture Notes in Computer Science*, pages 79–92. Springer, 2003.

[23] R. Zuccherato. Using a pki based upon elliptic curve cryptography. Entrust white paper, 2003. `http://www.entrust.com/resources`.