J Cryptogr Eng (2012) 2:179–188
DOI 10.1007/s13389-012-0039-x

REGULAR PAPER

Analysis of performance versus security in hardware realizations of small elliptic curves for lightweight applications

Vladimir Trujillo-Olaya · Timothy Sherwood · Çetin Kaya Koç

Received: 17 August 2012 / Accepted: 20 August 2012 / Published online: 13 September 2012

Abstract In this paper, we report the results of a comprehensive study of the security level versus the execution performance (and resource requirements) for hardware implementations of small elliptic curves, particularly targeted for lightweight applications, such as RFID tags and sensor nodes. The case study was performed for small elliptic curves (41–163 bits) over GF(2^m), where finite field elements are represented using polynomial and Gaussian normal bases. The idea behind using elliptic curves in this range is that we obtain small implementations suitable for the mentioned applications, however, this would be at the cost of less security since the Elliptic Curve Discrete Logarithm Problem (ECDLP) would be easier to break, i.e., would require fewer resources and less time for such small curves. Therefore, one must investigate both sides of the coin: first, hardware resources to implement such elliptic curves and the resulting total execution time for a single point multiplication; second, hardware resources to break such a curve and the resulting cost in terms of a defined metric, such as the total amount devices or dollars to solve the ECDLP in a given time duration. Following this reasoning, we studied the hardware (FPGA) implementations of small elliptic curves and determined the amount of resources (number of ALUTs, MEMs, REGs, the duration of clock, the total number of clock cycles and the total execution time) needed for a single point multiplication operation. We also studied the security level of each one of these curves, based on an attack model an associated cost metric. Under our proposed attack model, which we believe is very innovative; we considered three different platforms, namely PC, FPGA, and cloud computing. Due to the complexity of Cloud Computing configurations, we considered two different performance instances, namely, small (low budget) and high performance (relatively high budget). We then calculated the amount of resources and the total amount of dollars needed to solve each particular ECDLP, under different assumptions. We believe the results of our study will allow designers to select the appropriate curve for each application and the device, based on the perceived (or real) threat models that device is operating and the performance requirements of the elliptic curve protocol, such as ECDH, ECDH, or ECIES.

Keywords ECC · ECDLP · Polynomial basis · Normal basis · FPGA · VHDL

The research described in this paper was conducted while the V. Trujillo-Olaya was visiting UCSB by a grant of Colciencias.

V. Trujillo-Olaya (✉)
Bionanoelectronics Research Group, Universidad del Valle, Cali, Colombia
e-mail: vlatruo@univalle.edu.co

T. Sherwood · Ç. K. Koç
University of California, Santa Barbara, USA
e-mail: sherwood@cs.ucsb.edu

Ç. K. Koç
Istanbul Şehir University, Istanbul, Turkey
e-mail: koc@cs.ucsb.edu

1 Introduction

Advances in wireless communications and sensor technologies allow for the development of wireless sensor networks. However, these networks are resource constrained in terms of energy, computation, and memory, and they are vulnerable to attacks due to the use of unsecure wireless communication channel and lack of infrastructure. In this case, lightweight cryptography is an important topic due to the trade-off between cost, security and performance. On the other hand, public key cryptosystems are slower than symmetric systems but they provide high levels of security in a network. In recent years, elliptic curve cryptosystems (ECC) have increased in many applications due to the use of smaller key size than other cryptosystems such as RSA and use less power. In this case, ECC allows for a compact implementation for a given level of security. In this work, small elliptic curve hardware implementations have been designed for lightweight applications. In addition, in order to show how secure the designed processors are, an attack model is shown for different key sizes and using different platforms: PC, FPGA and Cloud Computing (small and high performance instances). This paper is organized as follows. Section 2 shows some related works. In Sect. 3, the mathematical background on elliptic curves and finite fields is introduced. In Sect. 4, we describe the attack model for small elliptic curves, based on different platforms: PC, FPGA and cloud computing using small and high performance instances. In Sect. 5, synthesis results of the selected small elliptic curves are shown. Finally, in Sect. 6, the paper is concluded.

2 Related work

There has been an increasing research in lightweight applications because of its applications in sensor networks; some of the papers about small implementations of cryptographic algorithms are presented in this section. In [1], a low power ECC processor is described, which uses 6,718 gates in a 0.13 μm CMOS technology over GF(2^{131}), which is a good idea of the level of security, which is approximately 65 bits of real security. In this work, the consumed power is 30 μW and the operating frequency is 500 MHz. In [2], the author presents a Modular Arithmetic Logic Unit (MALU) that can be used for both RSA and curve-based cryptography, in this case, the field sizes for ECC and HECC are scalable. Another part of that work described embedded systems using hardware/software co-design. The author implemented RSA 1024 and ECC over an 160-bit prime field on the 8051 microprocessor and the ECC over GF(2^{163}) and HECC over GF(2^{83}) designs on the ARM microprocessor and another design is a high speed programmable crypto processor supporting ECC over GF(2^{571}) and HECC over GF(2^{283}) synthesized with 0.13 μm CMOS technology. For low-power implementation, ECC over GF(2^{131}) can be performed using a power consumption of 22 μW with a 500 kHz clock. In [3], a configurable library (TinyECC) written in nesC for ECC operations in wireless sensor networks is described, that can be flexibly configured and integrated into sensor network applications. They perform the evaluation on platforms like MICAz, TelosB, Tmote Sky and Imote 2. TinyECC includes elliptic curve parameters recommended by SECG. In this case, TinyECC uses elliptic curve cryptography defined over a 160-bit prime field, and implements schemes such as ECDH, ECDSA, and ECIES. In [4], the design of an ECC processor over the 192-bit prime field is shown. In this case, the ECC processor shows an area of 0.35 mm^2 using 180 nm CMOS technology at a frequency of 175 kHz for RFID applications. In [5], an ECC implementation is performed on the 8bit ATMEGA128 micro-controller, which is the heart of MICA2 platform. However, they chose an elliptic curve over GF(2^{113}) because it offered more security than GF(2^{109}). In that case, ECDLP over GF(2^{109}) was solved and it took 17 months. In this work, the authors implemented ECDH and ECDSA schemes. In [6], the authors present the design of an elliptic curve processor for RFID over GF(2^{131}), GF(2^{139}), GF(2^{163}) using 0.25 μm CMOS technology. They implemented an elliptic curve version of Schnorraes identification protocol. In [7], the authors are focused on processing unit of a sensor node. In this case microcontrollers are suitable for wireless sensor network environments. Hence the selection of the microcontroller depends on the applications, energy consumption, storage, speed, and external I/O ports. In this work, microcontrollers are classified by their capabilities: Weak microcontrollers are highly constrained; Heavy duty ones are very powerful microcontrollers and Normal ones are resource-constrained but powerful enough to hold a complex application. In [8], a comparison between three different PKC implementations is given: Rabins scheme, NTRUEncrypt architecture, and ECC architecture. For Rabins scheme the multiplier circuit consumes a chip area of less than 17,000 gates with an average power consumption of 148.18 μW. For NTRUEncrypt architecture, it takes up a chip area of less than 3,000 gates consuming less than 20 μW, while a highly parallelized variant with 84 arithmetic units uses up to 16200 gates and approximately 120 μW at 500 kHz. For ECC architecture over a prime field, where $p = (2^{101} + 1)/3$, the area is equivalent to 18,720 gates and consumes just under 400 μW of power at a clock frequency of 500 kHz. In [9], the implementation of NTRUEncrypt in ASIC standard cell logic is presented. It uses about 3,000 gates with an average power consumption of less than 20 μW. Also, the authors present an implementation of Rabin's Scheme in a chip area of less than 17,000 gates with its accompanying static power consumption of 117.5 μW. Both of them are working at the frequency of 500 kHz. In [10], the authors presented a review of selected lightweight cryptographic implementations on hardware and software for symmetric and asymmetric ciphers. Besides a hardware implementation of ECC over GF(2^{113}), GF(2^{131}), GF(2^{163}), and GF(2^{193}).

3 Mathematical background

Elliptic curve cryptosystems are widely used in modern information technologies. In this section, some mathematical fundamentals are presented.

3.1 Elliptic curves arithmetic over GF(2^m)

An elliptic curve E over the binary field GF(2^m), is defined by the Eq. 1,

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

where a and $b \in$ GF(2^m), $b \neq 0$. It is well known that the set of points $P = (x, y)$, where $x, y \in$ GF(2^m), that satisfy the equation, together with the point ∞, called the point at infinity serving as the identity, form an additive commutative group Ea,b.

3.2 Representation of elements of binary fields

The binary field GF(2^m) or characteristic two finite field contains 2^m elements and can be view as a vector space over GF(2) with dimension m. All field elements can be represented uniquely as binary vectors of dimension m. There is a variety of ways to represent elements in a binary finite field, depending of the choice of a basis for representation. Polynomial basis and normal basis are commonly used and supported by the NIST [12] and other standards.

3.2.1 Polynomial basis

Finite fields of order 2^m are called binary fields or characteristic two finite fields. One way to construct GF(2^m) is to use a polynomial basis representation: The elements of GF(2^m) are the binary polynomials of degree at most $m - 1$:

$$\begin{aligned}\mathrm{GF}(2^m) = a_{m-1}x^{m-1} + \ldots + a_2x^2 + a_1x + 1 :\\ a_i \in 0, 1, 0 \leq i \leq m - 1\end{aligned} \tag{2}$$

Let $p(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_2x^2 + p_1x + 1$ (where $p_i \in$ GF(2)) be an irreducible polynomial of degree m over GF(2). Irreducibility of $p(x)$ means that the polynomial cannot be factored as a product of binary polynomials with degree less than m. In [11], Seroussi shows different irreducible trinomials and pentanomials for different key lengths.

The field element is usually denoted by the bit string $(a_{m-1}a_{m-2} \ldots a_1a_0)$ of length m, thus the elements of $GF(2^m)$ can be represented by the set of all binary strings of length m. The multiplicative identity element '1' is represented by the bit string (00…01) while the additive identity element is represented by the bit string of all 0's.

Field operations: the following arithmetic operations are defined on the elements of GF(2^m) when using a polynomial basis representation with reduction polynomial $p(x)$:

Addition: If we define the elements $a, b \in$ GF(2^m) to be the polynomials $A(x) = \sum_{i=0}^{m-1} a_i x^i$, $B(x) = \sum_{i=0}^{m-1} b_i x^i$ respectively, then their sum is written:

$$S(x) = A(x) + B(x) = \sum_{i=0}^{m-1} (a_i + b_i)x^i \tag{3}$$

Where, $a = (a_{m-1}a_{m-2} \ldots a_1a_0)$ and $b = (b_{m-1}b_{m-2} \ldots b_1b_0)$ are elements of GF(2^m), then $a{+}b = c = (c_{m-1}c_{m-2} \ldots c_1c_0)$ where the bit additions in Eq. (3) $(a_i + b_i)$ are performed modulo 2.

Multiplication: finite field multiplication of two field elements, where $A(x) = \sum_{i=0}^{m-1} a_i x^i$, $B(x) = \sum_{i=0}^{m-1} b_i x^i$ and $C(x) = \sum_{i=0}^{m-1} c_i x^i$ can be carried out by multiplying $A(x)$ and $B(x)$ and performing reduction modulo $p(x)$ or alternatively by interleaving multiplication and reduction, then the multiplication is shown as follows:

$$(b(x)a_{m-1}x^{m-1} + \ldots + b(x)a_1x + b(x)a_0) \bmod p(x) \tag{4}$$

$$C(x) = \sum_{i=0}^{m-1} b(x)a_i x^i \mod p(x) \tag{5}$$

Inversion: if a is a nonzero element in GF(2^m), the inverse of a, denoted a^{-1} is the element $c \in$ GF(2^m) for which $a * c = 1$.

3.2.2 Normal basis

Normal basis representations of an element over the finite field GF(2^m) have the computational advantage that squaring an element can be done very efficiently. However multiplying different elements can be cumbersome, in general for this reason ANSI X9.62 specifies that Gaussian Normal Basis (GNB) should be used, thus the multiplication is both simpler and more efficient [12]. A normal basis for GF(2^m) is as follows: $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$, where $\beta \in$ GF(2^m) and any element $\alpha \in$ GF(2^m) can be written as follows:

$$a = \sum_{i=0}^{m-1} \beta^{2^i} \tag{6}$$

The type T of a GNB is a positive integer, and allows measuring the complexity of the multiplication operation with respect to that basis. Generally, the type T of smaller value allows making a more efficient multiplication. For a given m and T, the field GF(2^m) can have at most one GNB of type T. A GNB exists whenever m is not divisible by 8. Let m and T be positive integers. Then the type T of a GNB for GF(2^m) exists if and only if $p = Tm + 1$ is prime. Table 1 shows different irreducible trinomials and pentanomials for

Table 1 Type of GNB for different key lengths

m	Type
41	2
53	2
79	4
89	2
97	4
113	2
163	4

different key lengths. If $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ is a GNB in the finite field GF(2^m), then the element $a = \sum_{i=0}^{m-1} \beta^{2^i}$ is represented by the binary string $(a_{m-1}a_{m-2}\ldots a_1a_0)$, where $a_i \in \{0, 1\}$. In this case, the bit string of all 1's represents the multiplicative identity element. The bit string of all 0's represents the additive identity element. An important result for the arithmetic of the GNB is the Fermat's little Theorem. For all $\beta \in$ GF(2^m) so that:

$$\{\beta^2\}^m = \beta \tag{7}$$

This theorem is important to carry out the squaring of an element in the finite field GF(2^m).

Field operations: the following arithmetic operations are defined on the elements of GF(2^m), when using a normal basis representation. The following arithmetic operations are defined on the elements of GF(2^m), when using a GNB of type T:

Addition: If $(a_{m-1}a_{m-2}\ldots a_1a_0)$ and $(b_{m-1}b_{m-2}\ldots b_1b_0)$ are elements of GF(2^m), then $a+b=c=(c_{m-1}c_{m-2}\ldots c_1c_0)$ where $c_i = (a_i + b_i) \bmod 2$.

Squaring: Let $a = (a_{m-1}a_{m-2}\ldots a_1a_0) \in$ GF(2^m), then $a^2 = (\sum_{i=0}^{m-1} a_i\beta^{2^i})^2 = \sum_{i=0}^{m-1} a_i\beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1}\beta^{2^i}$ due to Fermat's little Theorem $\beta^{2^m} = \beta$; then $a^2 = (a_{m-2}\ldots a_1a_0a_{m-1})$, in this case, squaring is a simple rotation of the vector representation.

Multiplication: in order to perform a multiplication, first, it is necessary to build a function F(U,V) on inputs $U = (u_{m-1}u_{m-2}\ldots u_1u_0)$ and $V = (v_{m-1}v_{m-2}\ldots v_1v_0)$ as follows:

$$F(U, V) = \sum_{k=0}^{p-2} U_{j(k+1)}V_{j(p-k)} \tag{8}$$

From Eq. 8 the sub indexes j(k+1) and j(p−k) can be computed as is shown in Algorithm 1.

Algorithm 1 Computation of $J(k)$

```
Require: $U = (u_{m-1}u_{m-2}\ldots u_1u_0)$;
    $V = (v_{m-1}v_{m-2}\ldots v_1v_0)$
Ensure: $J(1), J(2), \ldots, J(p-1)$
1: $p \leftarrow Tm + 1$
2: Let $u$ be an integer having order $T$ modulo $p$
3: $w \leftarrow 1$
4: for $k = 0$ to $T - 1$ do
5:   for $i = 0$ to $m - 1$ do
6:     $J(n) \leftarrow i$
7:     $n \leftarrow 2n \bmod p$
8:     $w \leftarrow uw \bmod p$
9:   end for
10: end for
```

The computation of $J(k)$ needs only be performed once per basis. Then, the product $(c_{m-1}c_{m-2}\ldots c_1c_0) = (a_{m-1}a_{m-2}\ldots a_1a_0) \times (b_{m-1}b_{m-2}\ldots b_1b_0)$ can be computed. In this case, Algorithm 2 shows the GNB multiplication algorithm.

Algorithm 2 GNB multiplication

```
Require: $A = (a_{m-1}a_{m-2}\ldots a_1a_0)$
    $B = (b_{m-1}b_{m-2}\ldots b_1b_0)$
Ensure: $C = (c_{m-1}c_{m-2}\ldots c_1c_0)$
1: $U \leftarrow A$
2: $V \leftarrow B$
3: for $k = 0$ to $m - 1$ do
4:   $c_k \leftarrow F(U, V)$
5:   $U \leftarrow U << 1$
6:   $V \leftarrow V << 1$
7: end for
```

Inversion: if $a \neq 0$ and $a \in$ GF(2^m), the inverse of a, is the unique element $c \in$ GF(2^m) for which $ac = 1$, where $c = a^{-1}$. The algorithm used for inversion is based on the identity:

$$a^{-1} = a^{2m-2} = (a^{2^{m-1}-1})^2 \tag{9}$$

In [13], Itoh and Tsujii proposed a method that minimizes the number of multiplications to calculate the inversion, which is based on the following identities:

$$a^{2^{m-1}-1} = \begin{cases} (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}} a^{2^{\frac{m-1}{2}}-1} & m \text{ odd} \\ a(a^{2^{m-2}-1})^2 & m \text{ even} \end{cases} \tag{10}$$

3.3 Elliptic Curve Point Multiplication

This work uses the López and Dahab [14] point multiplication, which does not have any extra storage requirements and the same operations (doubling and addition points) are performed in each iteration of the main loop, then it potentially increases the resistance to timing attacks. In terms of finite field multiplication, the approximate cost of computing kP using López and Dahab algorithm is $6m + 20$, which is an efficient implementation of Montgomery's ladder method for computing kP on non-supersingular elliptic curves over GF(2^m). the Algorithm 3 is shown as followin next.

Algorithm 3 López-dahab point multiplication

```
Require: $k = (k_{t-1} \dots k_1 k_0)$, $k_{t-1} = 1$, $P(x, y) \in E(GF(2^m))$.
Ensure: $kP$
1:  $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$
2:  for $i = t - 2$ downto 0 do
3:    if $k_i = 1$ then
4:      $T \leftarrow Z_1, Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2, X_1 \leftarrow xZ_1 + X_1X_2TZ_2$
5:      $T \leftarrow X_2, X_2 \leftarrow X_2^4 + bZ_2^4, Z_2 \leftarrow T^2Z_2^2$
6:    else
7:      $T \leftarrow Z_2, Z_2 \leftarrow (X_1Z_2 + X_2Z_1)^2, X_2 \leftarrow xZ_2 + X_1X_2TZ_1$
8:      $T \leftarrow X_1, X_1 \leftarrow X_1^4 + bZ_1^4, Z_1 \leftarrow T_2Z_1^2$
9:    end if
10: end for
11: $x_3 \leftarrow X_1/Z_1$
12: $y_3 \leftarrow (x + X_1/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (x_2 + y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y$
```

4 Attack model for small elliptic curves

In this paper, small elliptic curves for lightweight applications are considered, this section will analyze the time to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP) for each curve as well as the amount of resources that are needed to break an ECC system using different platforms. The best algorithm to solve the ECDLP is Pollard's rho algorithm [15]. The method searches for a collision in a pseudo-random walk through the points in a curve. In this case, the expected number of iterations or steps to solve the ECDLP is: $\frac{\sqrt{\pi \times 2^m}}{2}$. Taking into account the equation for the number of steps to solve the ECDLP, the time in days to solve the ECDLP performed for one machine is:

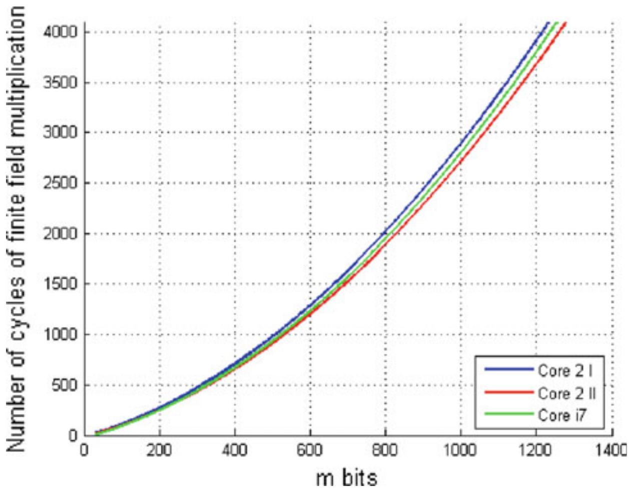

Fig. 1 Number of cycles of finite field multiplication based on [16]

$$T_{\text{brk-days}} = \frac{\frac{\sqrt{\pi 2^m}}{2}}{60 \times 60 \times 24 \times l} \tag{11}$$

where l is the number of iterations per second in one machine.

Due to one iteration or step is equivalent to the time to perform one point addition; in this work a point addition uses López Dahab projective coordinates which is the most popular way to represent points in elliptic curves over GF(2^m). In this case, a point addition operation costs $8M + 5S = 9M$ [16]. Where M and S are the time to perform one multiplication and one squaring over GF(2^m) respectively. Based on [17], the number of machines to carry out an attack in seconds in order to solve the ECDLP is shown as follows:

$$\frac{\frac{\sqrt{\pi 2^m}}{2}}{\frac{\text{freq. of machine}}{\frac{\text{no. of cycles per step}}{\text{no. of cores of the machine}}} \times \text{time to break in seconds}} \tag{12}$$

In this work, results from the López Dahab multiplication [16] are used and scaled to a lower field in order to obtain the number of cycles for finite field multiplication over small fields. In this case, three different processors are taken into account in order to calculate the number of cycles to perform one finite field multiplication. From a core 2 65 nm (core 2 I) the number of cycles to perform the finite field multiplication is determined by $0.0018\,\text{m}^2 + 1.0629\,\text{m} - 11.0869$; the number of cycles for a core 2 45 nm (core 2 II) is determined by $0.0017\ \text{m}^2 + 0.9929\ \text{m} - 17.447$ and the number of cycles for a core i7 45 nm is determined by $0.0018\ \text{m}^2 + 1.0198\ \text{m} - 25.543$ clock cycles, where m is the number of bits of the ECC. Figure 1 shows the number of cycles to perform one finite field multiplication for different key sizes [16].

Four assumed cases are taken into account in order to solve the ECDLP, each case is performed in a different platform such as personal computer (PC), FPGA, cloud computing using small instances (EC2 small) and Clod Computing using high performance clusters (EC2 medium).

4.1 PC model to solve ECDLP

The first case is when an attack is carried out using the Intel Core i7 45 nm, 4 cores, 3.07 GHz. In this case, making the assumption that a software implementation is performed to solve the ECDLP and based on [16], in which the authors describe an efficient software implementation of the finite field multiplication and present timings for several binary fields commonly used for ECC. In this case, this work takes into account that one iteration for solving the ECDLP is performed in 9 finite field multiplications.

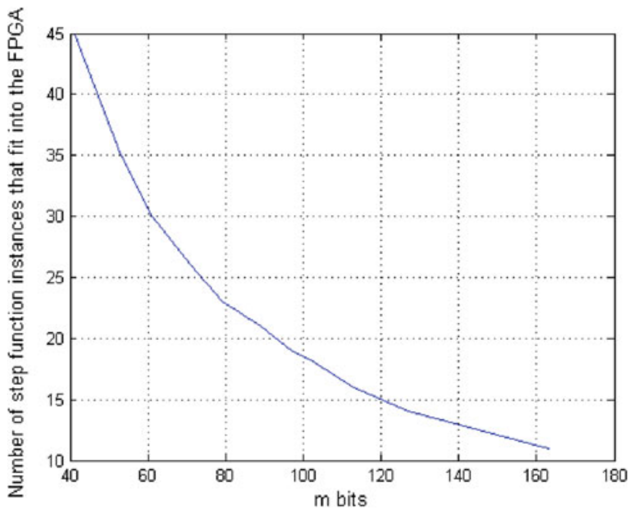

Fig. 2 Number of instances that fit into the FPGA in order to perform the step function

4.2 FPGA model to solve ECDLP

The second case is when several FPGAs are used in order to solve the ECDLP. In this case, according to [16], this work uses the Xilinx Spartan-3 XC3S5000-4FG676 FPGA, which is inexpensive. In order to perform the step function, the design will use one multiplier, one squarer and one Hamming weight counter. Making the assumption that one multiplication can be carried out in $\frac{m}{20}$ clock cycles, 120 MHz and the cost of one point addition is 9 multiplications [16], the number of cycles per iteration would be $9 \times \frac{m}{20}$. In addition, we can make several instances of the step function design into the FPGA taking into account that the number of instances is $\frac{74{,}880}{40 \times m}$, where 74,880 is the number of logic cells into the FPGA. Figure 2 shows the number of instances that fit into the FPGA.

4.3 Cloud computing for small instances model to solve ECDLP

The third case is when an attack is carried out using cloud computing using one small standard on demand instances, which provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. In this case, 1 h of use in Linux costs US $0.080.

4.4 Cloud computing for medium instances model to solve ECDLP

The fourth case is when an attack is carried out using cloud computing for medium computing instances which provide high CPU resources and one instance has 33.5 EC2 Compute Units (2 × Intel Xeon X5570, quad-core Nehalem architecture), One EC2 Compute Unit provides the equivalent

Table 2 Number of cycles per iteration and the time to solve the ECDLP

m	No. of cycles per iteration				Time to solve ECDLP using one instance			
	PC	FPGA	EC2 small	EC2 medium	PC	FPGA	EC2 small	EC2 medium
41	180	19	324	243	76.98 ms	207 ms	354 ms	266 ms
53	306	24	459	360	8.38 s	16.76 s	32.14 s	25.22 s
61	396	28	549	450	4.03 months	5.18 months	10.22 months	8.35 months
71	504	32	666	558	1.96 h	3.12 h	6.48 h	5.52 h
79	603	36	765	648	1.56 days	2.39 days	5.08 days	4.3 days
89	720	41	882	765	59.84 days	87.19 days	187.56 days	162.6 days
97	819	44	981	855	2.98 years	4.01 years	9.4 years	7.96 years
103	891	47	1,062	927	25.97 years	35.05 years	79.2 years	69.13 years
113	1,017	51	1,197	1,053	947.94 years	1,216 years	2,849 years	2,512 years
127	1,206	58	1,377	1,233	1.43×10^5 years	1.76×10^5 years	4.19×10^5 years	3.75×10^5 years
163	1,701	74	1,890	1,710	5.31×10^{10} years	5.91×10^{10} years	1.51×10^{11} years	1.36×10^{11} years

Table 3 Number of devices needed to solve the ECDLP

No. of devices

m	Time to solve ECDLP = 1 min				Time to solve ECDLP = 1 h			
	PC	FPGA	EC2 small	EC2 medium	PC	FPGA	EC2 small	EC2 medium
41	1	1	1	1	1	1	1	1
53	1	1	1	1	1	1	1	1
61	1	1	6	2	1	1	1	1
71	30	8	200	42	1	1	4	1
79	564	150	3,661	776	10	3	62	13
89	21,546	5,979	1.35×10^5	29,284	360	100	2,251	489
97	3.92×10^5	1.13×10^5	2.40×10^6	5.23×10^5	6,536	1,892	40,055	8,728
103	3.41×10^6	1.02×10^6	2.08×10^7	4.54×10^6	56,881	17,059	3.46×10^5	75,700
113	1.24×10^8	4.00×10^7	7.50×10^8	1.65×10^8	2.07×10^6	6.6105	1.25×10^7	2.75×10^6
127	1.89×10^{10}	6.65×10^9	1.10×10^{11}	2.47×10^{10}	3.15×10^8	1.1×10^8	1.84×10^9	4.12×10^8
163	6.99×10^{15}	2.83×10^{15}	$3{,}97 \times 10^{16}$	$8{,}99 \times 10^{15}$	1.16×10^{14}	4.71×10^{13}	6.62×10^{14}	1.49×10^{14}

Table 4 Funding needed to solve the ECDLP

m	Funding needed to break an ECC in 1 min				Funding needed to break an ECC in 1 h			
	US \$2,500	US \$100	US \$0.080	US \$1.60	US \$2500	US \$100	US \$0.080	US \$1.60
	1 PC	1 FPGA	1 EC2 small	1 EC2 medium	1 PC	1 FPGA	1 EC2 small	1 EC2 medium
41	2,500	100	0.080	1.60	2,500	100	0.080	1.60
53	2,500	100	0.080	1.60	2,500	100	0.080	1.60
61	2,500	100	0.480	3.20	2,500	100	0.080	1.60
71	75,500	800	16	67.20	2,500	100	0.32	1.60
79	1.41×10^6	15,000	292.88	1,241.60	25,000	300	4.96	20.8
89	5.38×10^7	5.97×10^5	10,800	46,854	9×10^5	10,000	180.08	782.4
97	9.8×10^8	1.13×10^7	1.92×10^5	8.37×10^5	1.63×10^7	1.89×10^5	3,204.04	13,965
103	8.53×10^9	1.02×10^8	1.66×10^6	7.26×10^6	1.42×10^8	1.7×10^6	28,800	1.21×10^5
113	3.11×10^{11}	4.00×10^9	6×10^7	2.64×10^8	5.19×10^9	6.6×10^7	1.00×10^6	4.4×10^6
127	4.73×10^{13}	6.65×10^{11}	8.8×10^9	3.95×10^{10}	7.88×10^{11}	1.1×10^{10}	1.47×10^8 years	6.59×10^8
163	1.74×10^9	2.83×10^{17}	3.17×10^{15}	1.43×10^{16}	2.91×10^{17}	4.71×10^{15}	5.29×10^{13}	2.39×10^{14}

CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor [19]. In this case, 1 h of use in Linux costs US \$1.60.

Table 2 shows the number of cycles per iteration that are needed to perform the step function as well as the time to solve the ECDLP and using different platforms such as personal computer (PC), FPGA, cloud computing for small instances (EC2 small) and high performance instances using clusters (EC2 medium). From Table 4, the time to solve the ECDLP when $m = 71$ is for one PC 1.9 h; for an FPGA is 3.12 h, for cloud computing using one small instance is 6.48 h and for cloud computing using one instance is 5.52 h. Table 3 shows the number of devices needed to solve the ECDLP in 1 min and 1 h. In this case, the FPGAs column uses the lowest amount of devices and the CC small column shows the biggest amount of instances. Table 4 presents funding needed to solve the ECDLP in 1 min and 1 h. In this case, the cloud computing for small instances column shows the lowest budget to solve the ECDLP. The amount of money that are needed in order to break an ECC in 1 min and 1 h using different platforms (PC, FPGA, instances of cloud computing) for different key sizes is shown in Fig. 3. In this case, personal computer to solve the ECDLP is an expensive option instead of cloud computing for small instances which shows the cheapest option to solve the ECDLP in 1 min and 1 h. An strong comparison of this work with some cryptanalitic

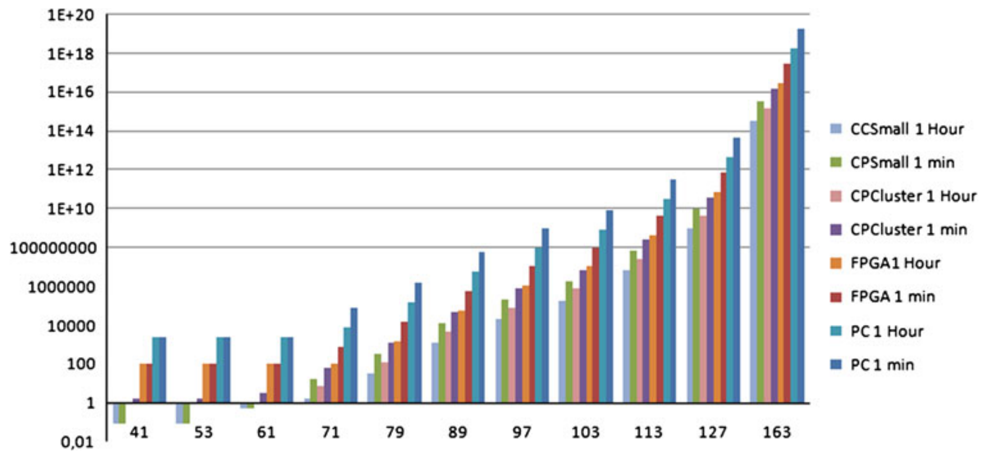

Fig. 3 Funding needed to solve the ECDLP for different platforms and key sizes

strength of ECC on EC2 presented in [18] would not be fair, however, the results presented in our work are very closer to those presented in [18].

5 Place and Route results for hardware implementations over small elliptic curves

In this section, we present place and route results of our elliptic curves hardware implementations over small fields. In this case, the processors were synthesized on the FPGA EP3SE50F780C2 using Quartus II. In this case, the FPGA resources are: ALUTs 38000, Registers 38000 and memory bits 5455872. Table 5 shows the place and route results for elliptic curves over polynomial basis. Table 6 shows place and route results for elliptic curves using GNB. From Tables 5 and 6, it is possible to note that the performance for the designed processors present slight differences. In this case, for the next figures and tables, this paper will use the data from Table 5. The budget needed to solve the ECDLP in one minute using different platforms against the time-area product of ECC processors implemented on FPGA is shown in Fig. 4. In this case, the time-area of an ECC is linear while the budget needed to solve the ECDLP is growing in an exponential way.

Figure 5 shows the number of machines and the budget needed to solve the ECDLP using different platforms, PC, FPGA, and Cloud computing with small and cluster instances. In this case the Cloud computing using small instances needs more instances but uses less amount of funding.

6 Conclusions

This work presents hardware implementations of small elliptic curves using Gaussian normal and polynomial bases. We have synthesized the processors on the FPGA EP3SE50F7 80C2 and using the Quartus II software. The ECDLP for small elliptic curves was analyzed using Pollards Rho method and an approximation of the expected time is given as well as the amount of resources needed to solve the ECDLP. In addition, four different platforms are assumed to solve the ECDLP. Attack models taking advantage of special pur-

Table 5 Place and Route results for elliptic curves over polynomial basis

m	ALUTS	REG	MEM	Tclock (ns)	No. of cycles	kP [Texec (s)]
41	2,049	1,704	820	5.78	5, 576	32.2
53	2,551	2,093	1, 060	6.20	9, 116	56.5
61	2,808	2,363	1, 220	6.60	11, 956	78.9
71	3,158	2,696	1, 420	6.80	16, 046	109.1
79	3,440	2,958	1, 580	7.46	19, 750	147.3
89	3,958	3,289	1, 780	8.68	24, 920	216.3
97	4,227	3,547	1, 940	8.78	29, 488	258.9
103	4,405	3,750	2, 060	9.06	33, 166	300.4
113	4,766	4,081	2, 260	9.20	39, 776	365.9
127	5,226	4,555	2, 540	9.50	50, 038	475.3
163	6,536	5,725	3, 260	10.40	80, 685	839.1

Table 6 Place and route results for elliptic curves over gaussian normal basis

m	ALUTS	REG	MEM	Tclock (ns)	No. of cycles	kP [Texec (s)]
41	2,153	1,551	820	5.64	5,699	32.14
53	2,451	1,902	1,060	5.84	9,328	54.47
79	3,129	2,239	1,580	6.75	20,224	136.51
89	4,258	2,952	1,780	7.26	25,365	184.14
97	4,705	3,183	1,940	7.57	29,876	226.16
113	5,277	3,650	2,260	7.67	40,341	309.42
163	8,203	5,107	3,260	9.66	81,826	790.40

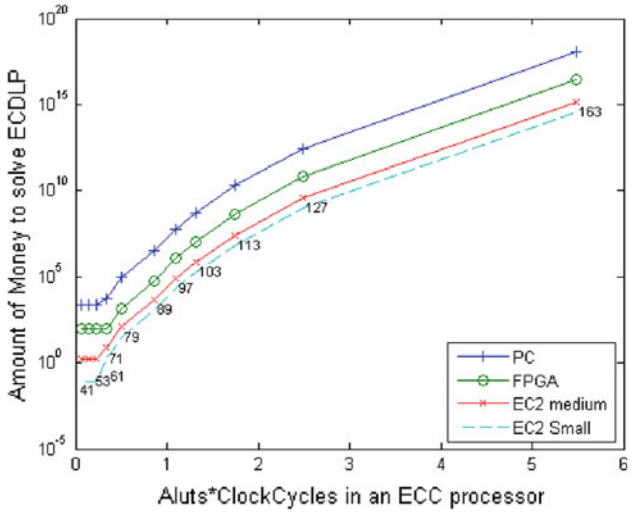

Fig. 4 Funding needed to solve the ECDLP against time-area product of ECC processors for different key sizes

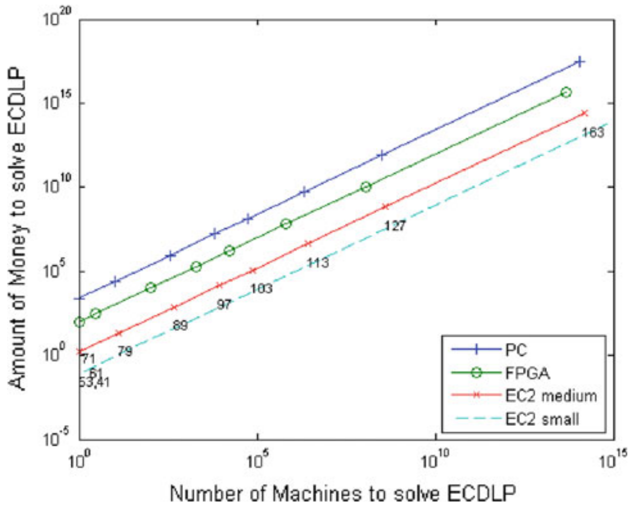

Fig. 5 Budget and number of machines needed to solve the ECDLP for different platforms

pose hardware using FPGAs have been considered as well as software using personal computers and cloud computing. In this case, the cheapest model in terms of money to solve the ECDLP is based on cloud computing using small instances but using several instances. Instead, the cheapest model attack in terms of amount of resources is the FPGA model. We believe the results of our study will be useful for designers to select the appropriate curve for each application and the device, based on the perceived threat models that device is operating and the performance requirements of the elliptic curve protocol, such as ECDH, ECDH, or ECIES, in the given environment and application.

References

1. Batina, L., Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Low-cost elliptic curve cryptography for wireless sensor networks. In: Proceedings of Third European Workshop on Security and Privacy in Ad Hoc and Sensor Networks, vol. 4357 of LNCS, pp. 6–17. Springer, Berlin (2006)
2. Sakiyama, K.: Secure design methodology and implementation for embedded public-key cryptosystems. PhD Thesis, Katholieke Universiteit Leuven (2007)
3. Liu, A., Ning, P.: Tiny ECC a configurable library for elliptic curve cryptography in wireless sensor networks. In: 2008 International Conference on Information Processing in Sensor Networks IPSN 2008, IEEE, pp. 245–256 (2008)
4. Wolkerstorfer, J.: Scaling ECC hardware to a minimum. In: Austrochip 2005 Mikroelektronik Tagung, Nikolaus Kerö und Peter Rössler, pp. 207–214 (2005)
5. Bla, E., Zitterbart, M.: Towards acceptable public-key encryption in sensor networks. In: The 2nd International Workshop on Ubiquitous Computing, ACM SIGMIS, pp. 88–93 (2005)
6. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: Public-key cryptography for RFID-tags. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications—Workshops (PerCom Workshops 2007), 19–23 March 2007, pp. 217–222. White Plains, New York (2007)
7. Roman, R., Alcaraz, C., Lopez, J.: A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. Mob. Netw. Appl. **12**(4), 231–244 (2007)
8. Gaubatz, G., Kaps, J., ztrk, E., Sunar, B.: State of the art in ultra-low power public key cryptography for wireless sensor networks. In: 2nd IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2005), pp. 146–150, Kauai Island (2005)
9. Gaubatz, G., Kaps, J., Sunar, B.: Public key cryptography in sensor networks—revisited. In: 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004), pp. 2–18 (2004)

10. Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., Uhsadel, L.: A survey of lightweight-cryptography implementations. IEEE Des. Test Comput. **24**(6), 522–533 (2007)
11. Seroussi, G.: Table of low-weight binary irreducible polynomials. Tech. Report, Computer Systems Laboratory, August (1998)
12. FIPS 186–2, Digital Signature Standard (DSS). http://csrc.nist.gov/publications/ps/ps186-2/ps186-2-change1.pdf
13. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. Inf. Comput. **78**, pp. 171–177 (1988)
14. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m) without precomputation. In: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems CHES '99, pp. 316–327 (1999)
15. Wiener, M.J., Zuccherato, R.J.: Faster attacks on elliptic curve cryptosystems. In: Proceedings of the Selected Areas in Cryptography SAC '98, pp. 190–200 (1999)
16. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Software implementation of binary elliptic curves: impact of the carry-less multiplier on scalar multiplication. Cryptology ePrint Archive, Report 2011/170 (2011)
17. Bailey, D.V., Batina, L., Bernstein, D.J., Birkner, P., Bos, J.W., Chen, H.C., Cheng, C.M., van Damme, G., de Meulenaer, G., Dominguez Perez, L.J., Fan, J., Gneysu, T., Gurkaynak, F., Kleinjung, T., Lange, T., Mentens, N., Niederhagen, R., Paar, C., Regazzoni, F., Schwabe, P., Uhsadel, L., Van Herrewege A., Yang, B.Y.: Breaking ECC2K-130. Cryptology ePrint Archive, Report 2009/541 (2009)
18. Kleinjung, T., Lenstra, A.K., Page, D., Smart, N.P.: Using the cloud to determine key strengths. Cryptology ePrint Archive, Report 2011/254 (2011)
19. http://aws.amazon.com/ec2/instance-types/. Accessed Nov 2011