Collision Search for Elliptic Curve Discrete Logarithm over GF(2^m) with FPGA

Guerric Meurice de Dormale*, Philippe Bulens**, and Jean-Jacques Quisquater

UCL DICE/Crypto Group, Place du Levant 3, B-1348 Louvain-La-Neuve, Belgium.
{gmeurice,bulens,quisquater}@dice.ucl.ac.be

Abstract. In this last decade, Elliptic Curve Cryptography (ECC) has gained increasing acceptance in the industry and the academic community and has been the subject of several standards. This interest is mainly due to the high level of security with relatively small keys provided by ECC. Indeed, no sub-exponential algorithms are known to solve the underlying hard problem: the Elliptic Curve Discrete Logarithm.
The aim of this work is to explore the possibilities of dedicated hardware implementing the best known algorithm for generic curves: the parallelized Pollard's ρ method. This problem has specific constraints and requires therefore new architectures. Four different strategies were investigated with different FPGA families in order to provide the best area-time product, according to the capabilities of the chosen platforms. The approach yielding the best throughput over hardware cost ratio is then fully described and was implemented in order to estimate the cost of an attack. Such results should help to improve the accuracy of the security level offered by a given key size, especially for the shorter parameters proposed for resource constrained devices.

1 Introduction

Since their introduction in cryptography in 1985 by Neal Koblitz [20] and Victor Miller [26], elliptic curves have raised increasing interest. This rich mathematical tool has been used to set up new asymmetric schemes able to compete with the well established RSA. Such schemes allow many useful functionalities like digital signature, public key encryption, key agreement, ... For those needs, Elliptic Curve Cryptography (ECC) is indeed an attractive solution as the provided public key scheme is currently one of the most secure per bit.

The underlying hard problem of ECC is the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP). Let $E(\mathbb{F})$ be an elliptic curve over a finite field $\mathbb{F}$ and let P be a point of $E(\mathbb{F})$. For any point $Q \in \langle P \rangle$ (the subgroup generated by P), the problem is to determine the integer k, $0 < k < n$, (with n the order of P) verifying $k \cdot P = Q$.

In day-to-day life utilizations of public key cryptography, choosing security parameters is a major concern[1]. Knowing the difficulty of solving ECDLP in-

* Supported by the Belgian fund for industrial and agricultural research (FRIA)
** Supported by Walloon Region / ALIS - First Europe Program
[1] A platform gathering current recommended key sizes can be found in [10].

stances is therefore essential to reach good trade-offs between security and computation power. Until 2006, the only released attacks were performed on general purpose processors. Hardware platforms can obviously be lower power and faster than software ones but the cost improving factor, for given $\mathrm{GF}(2^m)$ problems, is currently unknown. The aim of this work is therefore to evaluate the complexity and the cost of solving ECDLP by means of dedicated hardware. Such results should help to improve the accuracy of the security level offered by a given key size, especially for small key proposed for resource-constrained devices [32].

This work focusses on curves over $\mathrm{GF}(2^m)$ instead of $\mathrm{GF}(p)$. Indeed, $\mathrm{GF}(2^m)$ arithmetic is well suited for hardware platforms (addition is a simple bitwise xor) while software platforms are optimized for $\mathrm{GF}(p)$. A higher improvement factor over software solutions is therefore expected.

The framework here is a general attack on curves over $\mathrm{GF}(2^m)$, independently of the representation of the underlying group. Specific attacks, like MOV [23], Pohlig-Hellman [22] and the exploitation of weak fields $\mathrm{GF}(2^{mn})$ [9, 24] are not handled. For general attacks, the methods of interest are Pollard's ρ [15] or Shanks' Baby-step Giant-step [5]. The algorithm used in this work is the parallelized ρ method of van Oorschot and Wiener [38] with Teske's observations [37].

For the hardware platform, FPGAs instead of ASICs were chosen. Indeed, today's FPGAs got rid of most of the electrical and thermal problems they were suffering from in the last half decade. Prices for large FPGA devices are now also very affordable. As a result, its flexibility and performance over cost (for low volume) makes FPGAs an attractive solution. In particular, the FPGA-based COPACOBANA engine [17] was used for the cost assessment of this work.

Existing ECC architectures cannot be easily tailored to meet the specific requirements of the problem (cf. 4.2). New architectures with new trade-offs have therefore to be considered. In this work, four kinds of processors for solving ECDLP were investigated on different FPGA device families: *tiny* and *small* with low area requirements for low-cost FPGAs, *medium* for low-cost FPGAs and *large* for high-performance FPGAs. The aim is to provide architectures with different area-time complexity in order to best fit the capabilities of the chosen platforms. As a result, a high-performance device is not reduced to the sum of small devices. For that purpose, an original approach based on a prior theoretical analysis of area requirements of algorithms was used. This method allows selecting best options before implementations. Then, based on preliminary implementation results, the best processor is selected and fully described. Finally, timings of a software-based implementation are used to compare the performance-cost ratio of hardware platforms and general purpose processors.

This paper is structured as follows: Section 2 deals with previous attempts for solving ECDLP. Section 3 reminds the mathematical background of elliptic curves. Then, Section 4 explains the algorithm and improvements used for the collision search. The description of the architecture of the whole system stands in Section 5. After that, algorithms necessary for the arithmetic on elliptic curve are studied in Section 6. Based on this theoretical analysis, the four kinds of processors are presented in Section 7. Afterwards, the most efficient processor is fully described in Section 8. The hardware results and cost assessments are presented in Section 9 and finally, conclusions are given in Section 10.

2 Previous work

Until now, the hardest Certicom challenges [6] solved were done on 109-bit fields using general purpose processors. The ECC2-109 challenge was solved in 2004 by the team of Chris Monico. The effort required 2600 computers and took 17 months. The gross CPU time used was estimated equivalent to that of an Athlon XP 3200+ working nonstop for about 1200 years [6]. Those results suggest the use of dedicated hardware for attacking higher fields like for ECC2-131.

A survey about hardware for attacking ECC, based mainly on results of [11] about GF(p) curves can be found in [29].

A rough estimation about using dedicated hardware for solving ECDLP on curves over GF(2^{155}) with elements represented on an Optimal Normal Basis (ONB) can be found in [38]. Their aim was to show that the setup of [1] was insecure: that a largest prime factor with a size of only 2^{120} was insufficient. With a budget of \$10 million for their 1.5 μm ASICs clocked at 40 Mhz, the authors estimated that solving this particular ECDLP should only take 32 days[2].

Unlike this work, we focus on polynomial basis with low weight irreducible polynomials (as those recommended in standards like [27, 32]). Further analysis of ONB-based systems is left for another work.

The first results about concrete hardware implementation were presented in [11] for GF(p) and in [4] for GF(2^m). Compared to the (unpublished) results presented in [4], this work completes the study in many aspects. In particular, in addition to a *large* processor for high-performance FPGAs, *small* processors for low-cost FPGAs are studied. Moreover, power consumption and performance analysis in terms of the throughput-hardware cost ratio are given.

3 Mathematical Background

Let $p(z) \in \mathrm{GF}(2)[z]$ be an irreducible polynomial of degree m generating the field GF(2^m). A non supersingular elliptic curve E over GF(2^m) using affine coordinates can be defined as the set of solutions of the reduced Weierstraß equation:

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

where $a, b \in \mathrm{GF}(2^m)$, $b \neq 0$, together with the point at infinity $\mathcal{O}$.

The inverse of point $P = (x_1, y_1)$ is $-P = (x_1, x_1 + y_1)$. In affine coordinates, the sum $P + Q$ of points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ (assuming that $P, Q \neq \mathcal{O}$ and $P \neq -Q$) is point $R = (x_3, y_3)$ where:

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = \lambda \cdot (x_1 + x_3) + x_3 + y_1 \end{cases} \quad \text{with } \lambda = \begin{cases} \frac{y_1+y_2}{x_1+x_2} & \text{when } P \neq Q \\ \frac{y_1}{x_1} + x_1 & \text{when } P = Q \end{cases}$$

[2] However, the processor should be used in affine mode instead of projective (cf. 4.2).

Those modular arithmetic computations involve an expensive division, a multiplication, a squaring and several additions. When performing a scalar multiplication, the inversion is usually deferred to the end of the whole computation by using different coordinate systems. However, as an invariant is required while solving ECDLP, the use of affine coordinates is cheaper (cf. 4.2).

For a thorough description of elliptic curves, the reader is referred to [3].

4 Collision Search

The main algorithms for solving a generic ECDLP are Pollard's ρ [15] and Shanks' Baby-step Giant-step [5] methods. However, Pollard's ρ has the great advantage of requiring only a little amount of memory. The parallelized version of this method, with distinguished points, was chosen for this work as it is the best known algorithm to solve a generic ECDLP instance.

4.1 Pollard's ρ Algorithm

The basic idea to solve DLP is to walk in the group as randomly as possible until a collision is found, i.e. once a group element is reached twice, coming from different ways. The algorithm shown here is a general adaptation of Pollard's method to the case of the ECDLP.

To recover the unknown k, with $Q = k \cdot P$, a random chain is initialized by a point $R_0 = c_0 \cdot P + d_0 \cdot Q$. Then, the following function is iteratively applied:

$$R_{i+1} = \begin{cases} R_i + M_u & \text{when } R_i \in T_u \\ 2 \cdot R_i & \text{when } R_i \in T_v \end{cases}$$

for some point $M_u = e_u \cdot P + f_u \cdot Q$ with e_u, f_u randomly chosen. T_u (resp. T_v) is the set of partitions for which an addition (resp. doubling) is performed. In order to solve the DLP, c and d have to be updated accordingly. Following the birthday paradox, the probability to find $R_i = R_j$ increases, leading to the solution k:

$$c_i P + d_i Q = c_i P + d_i kP = c_j P + d_j kP = c_j P + d_j Q$$
$$k = \frac{c_j - c_i}{d_i - d_j} \bmod n$$

An appropriate number of T_u and T_v partitions has to be chosen. Teske's experimental results [37] suggest that setting $u \geq 20$ and $v = 0$ yields performances close to the optimal "true" random walk. This is particularly interesting as it allows the point addition to be solely implemented. In practice, a power of 2 is a convenient choice for u: a small chunk of bits of the point can be regarded as the index of the partition.

4.2 Improvements

Distinguished Points: The concept of Distinguished Points (DPs), attributed to Rivest by Denning [7], can be included to improve the overall efficiency of the algorithm. A distinguished point criterion, or property, is chosen and each computed point satisfying this criterion is stored. The collision search is now limited to a comparison between the distinguished points.

However, a given point can be represented in many ways using different coordinate systems. Without an invariant in point representation, DP criterion cannot be checked. Moreover, chains do not merge after a collision, preventing its detection while reaching a DP. To our knowledge, there is no invariant computationally cheaper than a modular division. As a result, affine coordinates are more efficient as point representation in this system is unique.

A well chosen DP criterion[3] will ensure that enough DPs are encountered to limit the number of steps ($\frac{1}{\theta}$ term below), but not too much (as those DPs are stored). More precisely: if θ is the proportion of points in $\langle P \rangle$ having the DP property, the expected number of elliptic curve operations before a collision of DP is observed is $\sqrt{\frac{\pi \cdot n}{2}} + \frac{1}{\theta}$, or $\frac{1}{C} \cdot \sqrt{\frac{\pi \cdot n}{2}} + \frac{1}{\theta}$ with C computers (cf. below).

To deal with the case where a chain falls into a loop without DP, chains exceeding 20 times the average length are dropped. This corresponds to a proportion of $(1-\theta)^{20/\theta} \approx e^{-20}$ chains with a waste of work of $20\, e^{-20} < 5\, 10^{-8}$[38].

Parallelization: Using a large number of computers, van Oorschot and Wiener [38] showed that it was interesting to make them share their knowledge. The collision search can therefore be viewed as a search between several chains. The complexity is reduced by the full number of computers C taking part in the collision search. Resorting to DP is particularly useful with the parallelization technique.

Negation and Frobenius Map: Other improvements, like negation map and frobenius map (for Koblitz curves) allow limiting the collision search to a fraction of the space [39]. They are not currently tackled and are left for another work.

5 Collision Search Architecture

The global collision search architecture is made of a server (e.g. a PC) and an arbitrary number of clients (for instance FPGAs). The hardware clients take care of the computationally intensive part of the work, namely the random walks on the elliptic curve. The software server handles the low-throughput operations: computation and dispatching of the starting points (SPs), recovery of the DPs, check for their correctness and sorting in order to find a collision. A software-based platform is a natural choice as it can provide the resources to sort the DPs and a huge memory to store them.

5.1 Software Program

The software is organized as follows: python language is used as the high level programming language and wraps, with swigwin [36], the low level C/C++ functionalities. The C++ modules use NTL [28] and deal with the number theory aspects while the C modules handle the communications. The performances of those modules are not critical.

[3] As pointed out in [33], some DP criteria lead to inefficient attacks on $GF(2^m)$.

5.2 Hardware Circuit

Each hardware client embeds a communication interface (basically a FIFO), a main controller and some numbers of elliptic curve processors with their own communication buffers (EC-μP). A ring communication topology is the most scalable approach, especially for a set of many small EC-μP on a large platform.

For the communication, each data set is made of a header, the x- and y-coordinates of a point and its c and d components in the P, Q basis. The header is used to determine the address of the EC-μP and the kind of data carried (either SP or DP). A more detailed view of an EC-μP is given in Fig. 1.

During the initialization phase, the data are sent from the main controller to the first EC-μP which collects one SP for each chain. Additional SPs are stored to allow starting a new chain when it exceeds the allowed length ($20/\theta$). SPs are deserialized to feed FIFOs in order to load RAMs of the *Point Update* and *Coeff Update* computing units. In most architectures, the throughput of the *Coeff Update* unit is sufficiently low to work on w-bit data (which avoids a specific (de)serialize circuit). When a DP is found, the point and its coefficients are stored in other FIFOs in order to serialize and output the data as soon as possible. In this case, the corresponding chain length is reset as well.

The default behavior of the EC-μP is to compute the point additions, update the coefficients and track the length of the chains. The number u of partitions (cf. Section 4.1) was set to 2^5 in order to directly map the 5 LSBs of the x-coordinate (called hash) to 32 partitions. For each point addition, this hash is sent through a FIFO to the *Coeff Update* unit in order to update the two coefficients of each chain accordingly. The *Point Update* unit also sent (through a FIFO) one bit for each chain to tell if a DP was found. When a chain exceeds the allowed length, its ID is sent to the *Point Update* unit through another FIFO.

6 Elliptic Curve Arithmetic

This section reviews different algorithms for the operations needed to compute an EC point addition (PA). Hardware implementations (e.g. [2]) usually focus on inversion-free coordinates as they are potentially more area-time efficient in the high-speed domain. Nevertheless, affine coordinates are cheaper and all the trade-offs have therefore to be reconsidered. Moreover, as a lot of area is potentially available in the targeted circuits, a lot of area-time trade-offs are achievable. In order to select the best algorithms, this analysis begins by reviewing their theoretical area requirements in terms of FPGA *slices*.

In this work, low-weight binary irreducible polynomials of degree m are considered. Such trinomials and pentanomials are recommended for implementations in [32, 27] and are used in challenges proposed in [6]. As sketched in Section 2, addition in GF(2^m) is a simple bitwise *xor* operation. Reconfigurable logic is used for the EC-μP, the polynomials can therefore be hardwired. As a result, each bit of the parallel modular reduction is computed with mainly 5-input *xor* gates for pentanomials and 3-input *xor* gates for trinomials. This leads to an inexpensive modular squaring circuit as the squaring itself is obtained by inserting a '0' bit between consecutive bits of the binary representation of the input polynomial [13]. Other relevant operations are multiplication and inversion/division.

6.1 Squaring

Most of the bits of a hardwired parallel squarer are computed with 2-input and 3-input *xor* gates for pentanomials and 1-input (no gate) and 2-input gates for trinomials. This corresponds to an area of $m/2$ slices. For circuits composed of parallel multiple squarers, logical simplifications occur. As a result, they are more area-time efficient than the iterative use of a single squarer.

6.2 Multiplication

For an iterative architecture, a digit-serial by parallel multiplier [34] is an attractive solution. With a D-bit digit, the computation needs $\lceil \frac{m}{D} \rceil$ cycles. The circuit is composed of a register for the parallel input, a shift register for the serial input and a registered $2D+1$-input sum of product (*and-xor* network) for the accumulated partial product and output. The modular reduction can be neglected. The following areas, in function of D, are required: $3m/2$ (1), $2m$ (2,3), $5m/2$ (4), $3m$ (5,6), $7m/2$ (7), $4m$ (8,9), $9m/2$ (10), $5m$ (11,12).

For a parallel multiplier, the sub-quadratic technique of Karatsuba is a popular choice (e.g. [30, 8]). The area depends on: the decomposition of m, the parameter used with the implementation tools (like *register balancing*) and the constraint on the operating frequency. A size of $(m/2)^2$ is achievable. More information about constructing such multipliers can also be found in [4].

6.3 Extended Euclidean Division

A divider based on the binary extended Euclidean algorithm [35] was presented in [40]. $2m-1$ cycles are required with an area of $2m$ slices [25]. A fully unrolled implementation would need the quite huge amount of $4m^2 - m^2/2$ slices. The $m^2/2$ term comes from the fact that half the data converge towards zero.

6.4 Montgomery Almost Inverse

The Montgomery almost inverse algorithm [16] could be used to save $m^2/2$ slices over a parallel[4] Euclidean-based circuit. Unfortunately, the cost of the domain transformation *almost* $\rightarrow$ *output* (cf. [12]) counterbalances this benefit. Moreover, an additional multiplication is needed to achieve the division.

6.5 Little Fermat's Theorem Inverse

Inverse could also be computed with little Fermat's theorem, stating that $\beta^{-1} = \beta^{2^m-2}$, with $\beta \neq 0 \in \mathrm{GF}(2^m)$. In order to minimize the number of multiplications, the chain technique of Itoh and Tsujii [14] can be employed. It needs m squarings and $\lfloor \log_2(m-1) \rfloor + \mathit{HammingWeight}(m-1) - 1$ multiplications.

[4] In the context of this paper, an iterative circuit cannot be improved by this method.

6.6 Montgomery trick

As inverters are usually more expensive than multipliers, the Montgomery trick [5] could be used. If the throughput of multiplier(s) is increased by a factor $3t-3$ $/4t-3$, the throughput of an inverter/divider can be reduced by a factor t. More memory is also required to store the different inputs and temporary data.

7 Elliptic Curve Processors

In order to best fit the capabilities of different FPGA families, several kinds of architectures are considered. The aim is to achieve the best area-time complexity on a given platform, resulting in a fair throughput/hardware cost comparison. For instance, a big high-performance device is not reduced to the sum of small devices: the high internal data bandwidth is exploited. For each of the four design principles, the algorithms of the architecture are selected using the theoretical area requirements provided in Section 6. The cost of control, RAMs, ROMs and the interface also have to be taken into account.

To ease this analysis, the operating frequency is supposed to be the same for all architectures. For small architectures, the bottleneck is the control logic while the frequency is limited by place and route problems in big architectures.

To avoid data dependency problems within an EC-μP, several chains can also be used in parallel. This is important to keep arithmetic operators busy (like the critical divider). It is even required when pipelining is introduced within the operators. The drawback is the increase in memory requirements.

7.1 Tiny

The first idea is to build a small footprint ALU, able to compute all the GF(2^m) arithmetic functions. As the behavior of serial by parallel multiplication and Euclidean division only differs in the shift direction, hybrid circuits are achievable [18, 19]. The squaring can be translated into a multiplication and the addition made of small additional logic. Beside the ROM storing the points of the u partitions, 3 memory locations are required to store x_3, y_3 and a temporary variable. A small GF(p) unit for updating coordinates and the chain length is also needed. Unfortunately, considering the area of the whole EC-μP, theoretical results show this solution leads to an inefficient area-time product.

7.2 Small

Another approach is to allow several small parallel arithmetic units: a Euclidean divider/inverter (Div/Inv), a digit-serial by parallel multiplier (Mult) and a squarer (Sqr) for instance. A dedicated squarer is however not needed as Mult performs faster than Div. Two chains are used to never let the divider idle. This architecture requires therefore $2m+1$ cycles to compute 1 PA.

As the latency of the multiplier with $D=2$ is 4 times smaller than the divider, it could be worth it to use Montgomery trick. However, while the processing of 1 chain requires 1 Div and 2 Mults, 2 chains require 2 Divs and 4 Mults, or 1

Inv and 9 Mults using Montgomery trick[5] with $t = 2$ and $D = 5$. Four chains are used with this approach and require 12 ($\leq 2^4$) memory locations: 8 for the 4 points data and 2 for temporary variables. Roughly speaking, $3m/2$ slices can be theoretically saved (without taking into account the increase of the logic needed by the control machine). As a result, Montgomery trick could be interesting.

7.3 Medium

If more area is available, it could be interesting to use sub-quadratic multiplication algorithms in order to improve the area-time product. As sketched in Section 6, an inverter based on Itoh-Tsujii with parallel multipliers could perform better than a Euclidean-based divider. For this architecture, one parallel Karatsuba multiplier (Mult) with a multi-squarer (Msqr) is used for the inversion and the two multiplications of the PA algorithm (cf. Fig. 2). The inverter architecture and especially the principle of a Msqr unit is similar to [31]. The main difference lies in the purpose of the circuit: area-time efficiency is required here, not only speed. This is achieved by using several chains and pipelining the arithmetic units. The number of cycles needed to compute one PA is $2 + \lfloor \log_2(m-1) \rfloor + \mathit{HammingWeight}(m-1) - 1$ ($= 11$ for $m = 163$).

The number of parallel chains used is equal to 16: the pipeline depth of both the Mult (8) and the Msqr (8). To maximize the utilization of the Msqr, the two multiplications can be computed *during* the inversion. The computational requirements of Msqr can therefore be reduced as two multi-squaring can be both computed in two iterations. This requires doubling the number of chains.

As an inversion instead of a division is computed anyway, Montgomery trick could prove efficient in this case. For $m = 163$, the number of cycles becomes 8 for one PA (25 % improvement). Twice the number of chains is required. The architecture is also more complicated as there are more different operations. However, the computational requirements, and therefore the area, of Msqr can be reduced if the seven multiplications are computed *during* the inversion.

7.4 Large

If a large amount of area is available, a fully pipelined design can be implemented in order to reach the best area-time product. The circuit can indeed be data-driven and fully specialized. Moreover, economy of scale can also be achieved. The number of chains is simply equal to the number of register stages in the pipeline, resulting in a PA every clock cycle. In particular, Itoh-Tsujii technique is used in parallel with repeated squarers and parallel Karatsuba multipliers. The reader is referred to [4] for further information about this strategy.

8 Chosen Elliptic Curve Processor

Appropriate algorithms were chosen thanks to the theoretical area requirements of Section 6. The *tiny* processor was discarded on that basis. Now, based on preliminary implementation results, area and throughput of each solution are evaluated. Such numbers are reported in Table 3 together with prices of corresponding

[5] Throughput of the Mult and the Inv must be equalized to maximize the efficiency.

FPGAs. "Preliminary results" means here that all the control structures were not fully implemented. However, performances are based on working components (Mult, Div, Sqr, RAMs, ...) and behavior correctness was checked on software. Based on those estimates, the *medium* processor on a low-cost Spartan3E appears as the solution with the best throughput over hardware cost (device price). According to the author, those results are sufficient to focus only here on the *medium* architecture. Nevertheless, exact description and full implementation of other processors could be the subject of another work.

8.1 The Medium Processor

The *medium* architecture is presented in Fig. 2 while the scheduling for $m = 163$ is available as Table 4. The main components are a Mult, a Msqr and an ALU. 3 RAMs and a ROM are also available to store coordinates and variables. Shift registers (Delays) are used to render the latency independent from the computed operation. A buffer is used to accumulate the partial result of the inversion in Mult. The buffer of Msqr is employed when a multi squaring is performed in multiple iterations. A comparator is also employed to check the DP criterion.

Based on synthesis results for $m = 163$ on a Spartan3E-1600, it is clearly interesting to perform the 2 multiplications during the inversion: the area saving is 50% (22% of the FPGA) for the Msqr and the area loss is 45% (5% of the FPGA) for the memory. 32 chains are therefore used. The Montgomery trick was currently not fully implemented. Using this strategy (64 chains), the memory increase is 70% (8% of the FPGA) and surprisingly, no significant savings are measured for the Msqr. As a result, if there is enough remaining area in the FPGA, the Montgomery trick approach could be interesting.

The *Coeff Update* and *Chain Length Check* units serially process the data. They are simply made of 4 bRAMs (1 ROM and 2 RAMs for c, d and 1 for chain lengths) and a few logic for 16-bit modular adders, counters and comparators.

9 Hardware Results and Cost Assessment

In this section, hardware implementation results and cost assessment for attacking GF(2^m) ECDLP were achieved for 3 security levels recommended in [32].

9.1 Hardware results

All the hardware modules were written in VHDL and placed & routed on either XC3S1200E-4FT256 (21 US$) or XC3S1600E-5FG320 (33 US$) low-cost FPGAs[6]. Table 1 reports the different performances for a clock period constrained to 10 ns. For $m = 113$, two cores are embedded in the FPGA. Notice that bRAMs requirements depend on whether enough of such RAMs was available to implement 32 or $64 \times m$ memories. The throughput over FPGA cost is used to allow fair comparison with other FPGA platforms. The power consumption was also estimated using the *Xilinx Spartan-3E Web V8.1* power tool. The electricity cost for one year, assuming a price of 0.1 US$ per kWh, is also reported.

[6] FPGA costs are 2007/2008 Xilinx prices for 1000 devices.

Table 1. Medium EC-μP, Place and Route results from Xilinx ISE 9.1.

m	FPGA	Area [kSlices]	Area [bRAMs]	Freq. [Mhz]	Throughput [PA/s]	Thr./cost [PA/s\$]	Cons. [W]	Elec. price [\$/1 year]
113	S3E1600-5	13.9 (94%)	18 (50%)	100	$2 \times 10\ 10^6$	$6\ 10^5$	4.2	3.7
131	S3E1200-4	7.9 (91%)	21(75%)	100	$10\ 10^6$	$4.8\ 10^5$	3.2	2.8
163	S3E1600-5	10.9 (74%)	25 (69%)	100	$9.1\ 10^6$	$2.7\ 10^5$	3.8	3.3

9.2 Cost Assessments

It was planned to compare hardware with homemade software results. Unfortunately, our code based on the C++ NTL library [28] has poor performances compared to Certicom challenge results. Chris Monico's challenge results for GF(2^{109}) will therefore be used for further comparisons.

For the hardware cost assessment on FPGA, the COPACOBANA engine [17] was chosen. It embeds 120 low-cost Spartan3-1000 FPGAs. The Spartan3E-1600 FPGA selected for this work has twice the area of a Spartan3-1000 FPGA. Nevertheless, the price of both devices is the same (for 1000 parts) and their form factor is similar. The price of one COPACOBANA engine is therefore assumed to remain 10 kUS$. Concerning the power consumption, the xilinx web power tool suggests that the power consumption is doubled with the S3E FPGAs. As a result, doubling the power consumption of the original engine seems safe. Based on power consumption needed to attack DES with COPACOBANA (600 Watts), we will assume a power consumption of 1.2 kWatts for our application. Exact power consumption is left for further work.

For the cost assessment reported in Table 2, achievable performances are estimated through the expected running time (ERT). It is derived from performances of the architecture and expected number of group operations (ENO) before a collision occurs (cf. 4.2). For ENO, n is set as 2^m and θ as $2^{-m/3}$. The expenses for the power consumption were added for the total cost. For an attack during 1 year, it represents a 10^{th} of the purchase cost of the device.

Those results show that GF(2^{113}) is far from secure. For that attack, the number of required FPGA's is sufficiently small to overcome an ASIC-based solution. While the cost for the GF(2^{131}) attack is still tractable, an ASIC-based platform should be more cost-effective. For $m = 163$, a standard level security parameter, it is currently impossible to mount an FPGA-based attack.

Based on those results and [21], a rough ASIC extrapolation for a 90 nm CMOS techno using standard cells can be sketched[7]. The following parameters are assumed: area factor is 20, speed factor is 3.5, consumption factor is 14, die size of S3E-1600 is 2.5×2.5 mm and cost of one 300 mm wafer is 30 k$ (multiplied by two for overheads like packaging, cooling, ...). This rough estimation leads to a cost of $2.2\ 10^9$ \$ neglecting NREs for $m = 163$, which is still currently out of reach. Notice that half the price is for power consumption.

[7] Assuming the circuits are built to attack a given ECDLP instance many times, meaning that both the field size and the irreducible polynomial can be hardwired.

Another goal of this cost assessment is to compare hardware and software-based solutions. To solve the GF(2^{109}) challenge in 6 months, 2400 Athlon XP 3200+ were needed. By modifying the $m = 113$ result of Table 2, this problem is expected to be solved by one COPACOBANA engine in 6 months. Assuming a price of 150 $ for a computer and a consumption of 250 Watts, the purchase price ratio is **35**. This is a bit less than expected, provided those general purpose CPUs have no dedicated GF(2^m) ALU. From a power consumption prospect, the ratio is as big as **500**. Those results clearly justify the use of hardware.

Table 2. Cost assessments to solve different GF(2^m) ECDLP using COPACOBANA

m	ENO [#PA]	Thr. [#PA/s]	ERT [s]	ERT [s] on 1 Copa	#Copa for 1-year	Cost [US$] of power supply	Total cost [US$] 1-year
113	128 10^{15}	20 10^6	6.4 10^9	53.3 10^6	2	2.1 10^3	22.1 10^3
131	65.4 10^{18}	10 10^6	6.54 10^{12}	54.5 10^9	1728	1.8 10^6	19.1 10^6
163	4.23 10^{24}	9.1 10^6	471 10^{15}	4 10^{15}	125 10^6	131 10^9	1.4 10^{12}

As a GF(2^m) attack is supposed to perform better in hardware than a GF(p) one, it is interesting to compare them. From [11], the throughput for GF(p) and $k = 160$ (Table 1) is 93.6 kPA/s. Multiplying this result by 2 (to estimate performances on a S3E-1600) and comparing with the $m = 163$ result of Table 1, it appears that the throughput ratio is near **50**. For fairness purpose, notice that no parameters are hardwired in their architecture.

10 Conclusion

This work presented a thorough analysis of an FPGA-based solution to attack elliptic curve cryptosystems over GF(2^m). The provided results complete in many aspects the few previous studies. In particular, the performance-cost optimum for four different architectures was investigated on two different families of Xilinx FPGAs. The selected architecture, based on a low-cost Spartan3E FPGA, was described and implemented. Cost assessments were then performed on the basis of a cluster of low-cost FPGAs.

Compared to the software used to solve the GF(2^{109}) challenge, the hardware exhibits an improvement factor of 35 for purchase costs and 500 for power consumption. Considering current architectures to attack GF(p) and GF(2^m) instances, a throughput ratio of 50 was also measured. This follows the expectations as GF(2^m) arithmetic is particularly well suited for hardware platforms.

The lowest security level of the SECG standard, based on GF(2^{113}), was found to be *easily* breakable. However, even by taking the rough ASIC extrapolation, a standard security level like GF(2^{163}) stays out of reach of current attacks. Breaking those systems would require either far too much time given a moderate amount of money, or a prohibitive price to raise a solution in a reasonable amount of time. Given today's know-how and understanding of ECC, it is expected that current standard security keys do not suffer from any threat for the time being.

References

1. G.B. Agnew et al., An Implementation of Elliptic Curve Cryptosystems over $F_{2^{155}}$, *IEEE Journal on Selected Areas in Communications*, vol. 11(5), 804-813, 1993.
2. B. Ansari, M. Anwar Hasan, High Performance Architecture of Elliptic Curve Scalar Multiplication, *CACR Research Report 2006-01*, 2006.
3. I.F. Blake, G. Seroussi, N.P Smart, *Elliptic Curves in Cryptography*, London Mathematical Society, Lecture Notes Series 265, Cambridge University Press, 1999.
4. P. Bulens, G. Meurice de Dormale, J.-J. Quisquater, Hardware for Collision Search on Elliptic Curve over GF(2^m), *SHARCS*, Ecrypt Workshop, 2006.
5. H. Cohen, *A course in computational algebraic number theory*, Graduate Text in Mathematics, 138, Springer-Verlag, New York, 1993.
6. Certicom, `http://www.certicom.com`.
7. D.E. Denning, *Cryptography and Data Security*, Addison Wesley, 1982.
8. J. von zur Gathen, J. Shokrollahi, Fast arithmetic for polynomials over $\mathbb{F}_2$ in hardware, *IEEE ITW 2006*, pp. 107-111, 2006.
9. P. Gaudry, F. Hess, N. P. Smart, Constructive and Destructive Facets of Weil Descent on Elliptic Curves, *Journal of Cryptology*, vol. 15, no. 1, pp. 19-46, 2002.
10. D. Giry, P. Bulens, `http://www.keylength.com`.
11. T. Güneysu, C. Paar, J. Pelzl, Attacking elliptic curve cryptosystems with special-purpose hardware, *FPGA'07*, ACM/SIGDA, pp. 207-215, 2007.
12. A. A.-A. Gutub, *New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields GF(p) and GF(2^n)*, Ph.D. Thesis, 2002.
13. D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional computing, Springer, 2004.
14. T. Itoh, S. Tsujii, A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases, *Information and Computation*, vol. 78, pp. 171-177, 1988.
15. J.M. Pollard, Monte Carlo Methods for Index computation (mod p), *Mathematics of computation*, vol. 32, n143, pp. 918–924, July 1978.
16. B. S. Kaliski Jr., The Montgomery Inverse and its Applications, *IEEE Transactions on Computers*, vol. 44(8), pp. 1064-1065, August 1995.
17. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, M. Schimmler, Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker, *CHES 2006*, LNCS 4249, pp. 101-118, 2006.
18. C.H. Kim, S. Kwon, J.J. Kim et al., A New Arithmetic Unit in GF(2^m) for Reconfigurable Hardware Implementation, *FPL 2003*, LNCS 2778, pp. 670-680, 2003.
19. M.G. Kim, S.J. Yu, Y.S. Lee, J.S. Song, A Fast Hybrid Arithmetic Unit for Elliptic Curve Cryptosystem in Galois Fields with Prime and Composite Exponents, *IEICE Electronics Express*, vol. 1(1), pp. 13-18, 2004.
20. N. Koblitz, Elliptic curve cryptosystems, *Math. of computation*, vol. 48, pp. 203-209, 1987.
21. I. Kuon, J. Rose, Measuring the Gap Between FPGAs and ASICs, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 62, no. 2, Feb 2007.
22. A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer, 1993.
23. A. Menezes, T. Okamoto, S.A. Vanstone, Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field, *ACM Symp. Theory Computing*, pp. 80-89, 1991.
24. A. Menezes, E. Teske, A. Weng, Weak fields for ECC, *CT-RSA 2004*, LNCS 2964, pp. 366-386, 2004.
25. G. Meurice de Dormale, J.-J. Quisquater, Iterative Modular Division over GF(2^m): Novel Algorithm and Implementations on FPGA, *ARC 2006*, LNCS 3985, pp. 370-382, 2006.
26. V. Miller, Uses of elliptic curves in cryptography, *CRYPTO'85*, LNCS 218, pp. 417–426, 1986.
27. U.S. Department of Commerce/National Institute of Standards and Technology (NIST), Digital Signature Standard (DSS), FIPS PUB 182-2change1, 2000.
28. NTL : A Library for doing Number Theory, `http://www.shoup.net/`.
29. J. Pelzl, Exact Cost Estimates for Attacks on ECC with Special-Purpose Hardware, *Workshop on Elliptic Curve Cryptography - ECC 2006*.
30. F. Rodríguez-Henríquez, Ç.K. Koç, On Fully Parallel Karatsuba Multipliers for GF(2^m), *(394) Computer Science and Technology - 2003*, 2003.
31. F. Rodríguez-Henríquez et al., Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials, `http://eprint.iacr.org/2006/035.pdf`, 2006.
32. Certicom Research, SEC 2: Recommended Elliptic Curve Domain Parameters, v1.0, 2000.
33. N. P. Smart, A note on the x-coordinate of points on an elliptic curve in characteristic two, *Technical Report CSTR-00-019*, University of Bristol, December 2000.
34. L. Song, K. K. Parhi, Low energy digit-serial/parallel finite field multipliers, *Journal of VLSI Signal Processing*, vol. 19(2), pp. 149166, 1998.
35. J. Stein, Computational problems associated with Racah algebra, *Journal of Computational Physics*, vol. 1, pp. 397-405, 1967.
36. SWIG : Simplified Wrapper and Interface Generator, `http://www.swig.org/`.
37. E. Teske, On Random Walks for Pollard's rho method, *Mathematics of computation*, vol. 70, n. 234, pp. 809–825, 2000.

38. P. C. van Oorschot and M. J. Wiener, Parallel Collision Search with Cryptanalytic Applications, *Journal of Cryptology*, 12, pp. 1–28, 1999.
39. M. J. Wiener and R. Zuccherato, Faster Attacks on Elliptic Curve Cryptosystems, *Selected Areas of Cryptography – SAC'99*, Springer, LNCS 1556, pp. 190–200, 1999.
40. C.H. Wu et al., High-Speed, Low-Complexity Systolic Designs of Novel Iterative Division Algorithms in GF(2^m), *IEEE Transaction on Computers*, vol. 53(3), pp. 375-380, 2004.

A EC-μP Architectures

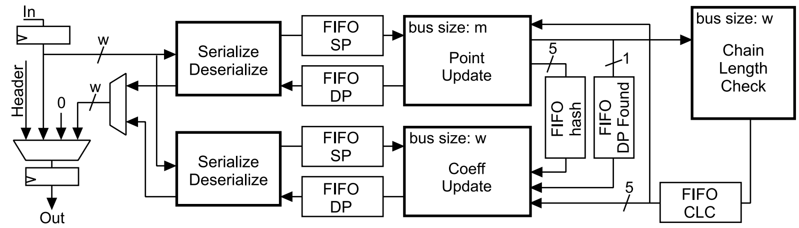

Fig. 1. Elliptic curve processors (EC-μP).

B Processors' Comparison Based on Preliminary Results

Table 3 reports the preliminary results for the different design strategies on a low-cost Spartan3E (XC3S1600E-4, 14 kSlices, 38$) and a high performance Virtex4 (XC4LX200-10, 89 kSlices, 2070$) FPGA. Those Xilinx's prices stand for 1000 devices. For the different extension fields, those results suggest that the most cost-effective choice is a *medium* architecture on low-cost FPGAs. The price of a big Virtex4 is dominated by die and yield costs. As architectures do not really benefit from V4 specific features, it seems inappropriate to solve ECDLP.

Table 3. Area and throughput estimations for each processor

		small		*medium*		*large*
FPGA		S3E1600	V4LX200	S3E1600	V4LX200	V4LX200
Area	$m = 79$	1.2		4.8		22.2
[kSlices]	$m = 109$	1.6		7.4		41.3
	$m = 163$	2.4		13.8		81.8
Frequency [Mhz]		100	250	100	250	250
Throughput/EC-μP		Freq./$(2m+1)$		$9\ 10^6$	$22.7\ 10^6$	$250\ 10^6$
#EC-μP		5 to 11	36 to 72	1 to 2	6 to 17	1 to 3
Through./cost	$m = 79$	182 (11)	55 (72)	478 (2)	187 (17)	362 (3)
[kPA/s$]	$m = 109$	96 (8)	30 (54)	239 (1)	120 (12)	242 (2)
	$m = 163$	40 (5)	13 (36)	239 (1)	66 (6)	120 (1)

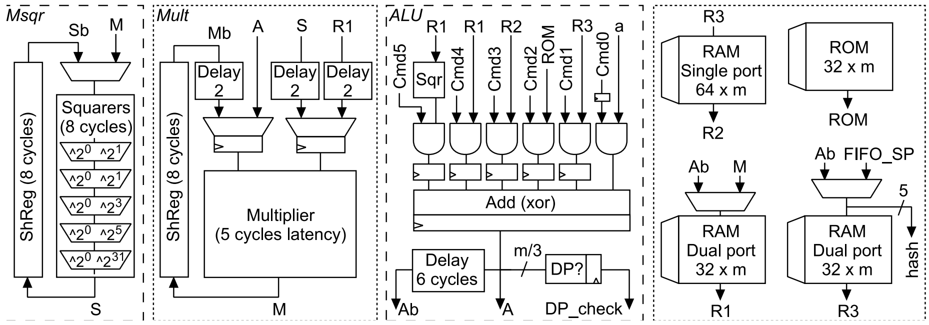

Fig. 2. Medium processor, *Point Update* unit (squarers for $m = 163$)

C The Medium Processor

The scheduling of the PA for $m = 163$ is presented in Table 4. The method is similar for other extension degrees. Each PA_i state represents 2 times 8 cycles. As Msqr follows Mult, its computations are delayed by 8 cycles. Write operations happen 8 cycles later as latency of arithmetic units is 8 cycles. Two passes are necessary to compute PAs. One prime on a variable means that data of one set of 16 chains are processed. No prime or two primes means the other set of 16 chains is treated. The role of those sets is exchanged at each pass. The a constant is the curve parameter (cf. Section 3).

Table 4. Scheduling of point additions for $m = 163$, without initialization

	S (Msqr)	M (Mult)	R1 ($32\times m$ RAM)	R2 ($64\times m$ RAM)	R3 ($32\times m$ RAM)	A,Ab (ALU)
PA_1	2Sqr(M)	A × R1 Mb ← M	$\mathrm{R1}_0$ $(x_1 + x_2)$	$\mathrm{R2}_0 \leftarrow \mathrm{R3}$	$\mathrm{R3}_0$ (x_3'')	Sqr(R1) $((x_1 + x_2)^2)$
PA_2	Sqr(M)	Mb × S Mb ← M	$\mathrm{R1}_1$ $(\sqrt{\mathrm{Inv}'})$ $\mathrm{R1}_1 \leftarrow \mathrm{Ab}$	$\mathrm{R2}_1 \leftarrow \mathrm{R3}$	$\mathrm{R3}_1$ (y_3'')	Sqr(R1) (Inv')
PA_3	5Sqr(M)	Mb × R1 Mb ← M	$\mathrm{R1}_0$ $(x_1 + x_2)$			
PA_4	10Sqr(M)	Mb × S Mb ← M				
PA_5	10Sqr(M)	Mb × S Mb ← M				
PA_6	Sb ← S 10Sqr(M)	A × R1	$\mathrm{R1}_1$ (Inv') $\mathrm{R1}_1 \leftarrow \mathrm{M}$	$\mathrm{R2}_1$ (y_1')		Add(R2,ROM) $(y_1' + y_2')$
PA_7	40Sqr(M)	Mb × S Mb ← M	$\mathrm{R1}_1$ (λ')	$\mathrm{R2}_0$ (x_1')	$\mathrm{R3}_0 \leftarrow \mathrm{Ab}$	Add(Sqr(R1), R1,R2,ROM,a)
PA_8	Sqr(M)	Mb × S Mb ← M				
PA_9	40Sqr(M)	Mb × R1 Mb ← M	$\mathrm{R1}_0$ $(x_1 + x_2)$ $\mathrm{R1}_0 \leftarrow \mathrm{Ab}$		$\mathrm{R3}_0$ (x_3')	Add(R3,ROM) $(x_3' + x_2')$
PA_{10}	Sb ← S 41Sqr(M)	A × R1	$\mathrm{R1}_1$ (λ') $\mathrm{R1}_1 \leftarrow \mathrm{M}$	$\mathrm{R2}_0$ (x_1')	$\mathrm{R3}_0$ (x_3')	Add(R2,R3) $(x_1' + x_3')$
PA_{11}		Mb × S	$\mathrm{R1}_1$ $(\lambda'(x_1' + x_3'))$ $\mathrm{R1}_1 \leftarrow \mathrm{M}$	$\mathrm{R2}_1$ (y_1')	$\mathrm{R3}_0$ (x_3') $\mathrm{R3}_1 \leftarrow \mathrm{Ab}$	Add(R1,R2,R3) (y_3')