On the Security of Elliptic Curve Cryptosystems against Attacks with Special-Purpose Hardware

Tim Güneysu, Christof Paar, Jan Pelzl

Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{gueneysu,cpaar,pelzl}@crypto.rub.de

Abstract

Since their invention in the mid 1980s, Elliptic Curve Cryptosystems (ECC) have become an alternative to common Public-Key (PK) cryptosystems such as, e.g., RSA. The utilization of Elliptic Curves (EC) in cryptography is very promising because of their resistance against powerful index-calculus attacks. For a similar level of security as RSA, ECC allows for efficient implementation due to a significantly smaller bit size of the operands. It is widely accepted that the only feasible way to attack actual cryptosystems is the application of dedicated hardware. In times of continuous technological improvements and increasing computing power, the question of the security of ECC against attacks based on special-purpose hardware arises.

This work presents an architecture and the corresponding FPGA implementation of an attack against ECC over prime fields. We describe an FPGA-based multi-processing hardware architecture for the Pollard-Rho method which is, to our knowledge, currently the most efficient attack against ECC. The implementation is running on a contemporary low-cost FPGA which allows for a much better cost-performance ratio than conventional CPUs. With the implementation at hand, a fairly accurate estimate about the cost of an FPGA-based attack can be given. We will project the results on actual ECC key lengths (160 bit and above) and estimate the expected runtimes for a successful attack. Since FPGA-based attacks are out of reach for actual key lengths, we provide estimates for an ASIC design.

As a result, we consider ECC over over prime fields to be far more secure than commonly believed. Based on our results, we estimate the effort to break ECC-163 with recent hardware architectures is much higher than that of RSA-1024. As a consequence, currently used elliptic curve cryptosystems are infeasible to break with available computational and financial resources.

Keywords: discrete logarithm, elliptic curve cryptosystem, cryptanalysis, Pollard's Rho, hardware, public-key, field programmable gate array.

1 Introduction

The Elliptic Curve Cryptosystem (ECC) was proposed independently by Neil Koblitz and Viktor Miller in 1985 [19, 15] and is based on the difficulty of the Diffie-Hellman Problem (DHP) in the group of points on an Elliptic Curve (EC) over a finite field. The DHP is closely related to the well studied Discrete Logarithm Problem (DLP). In contrast to the more efficient index-calculus attacks on the DLP over finite fields, ECC only allows for generic attacks such as Pollard's Rho method [22, 25]. This benefit yields much shorter underlying bit lengths for ECC (160...256 bit) compared to RSA or DLP in finite fields (1024...4096 bit) at an equivalent level of security [17].

Since its introduction, ECC has been extensively studied not only by the research community but also in industry. In particular, there are several standards involving ECC, such as the IEEE P1363 [13] standardization effort and the banking standards [2, 1]. All modern security protocols such as IPSec [14], SSL [10], TLS [6] use symmetric-key algorithms as well as public-key algorithms. Providing highly arithmetic-intensive public-key primitives is often challenging due to the high memory and power consumption. It is important to point out that ECC benefits from shorter operand sizes compared to, e.g., RSA or DSA, and is an attractive alternative.

The security of nearly all practical cryptosystems is based on computational assumptions. From a mathematical point of view, it is not impossible to reveal the secret information without the secret key: The ciphertext is encrypted such that it is infeasible to break with current cyptanalytical algorithms and computational resources. Due to technological improvements and the continuous increase of computing power, it becomes more and more important to encrypt confidential data with cryptographic systems at an appropriate level of security, reflected by an adequate choice of the key length to prevent attacks from becoming practical.

For RSA and DLP in finite fields, many publications address the issue of hardware-based attacks and provide security estimates based on proposals of architectures attacking such cryptosystems. For ECC however, no precise architecture for a hardware attack has been described yet. Cryptanalysis of ECC requires the same algorithmic primitives as the cryptosystem itself, namely point addition and point doubling which can be implemented very efficiently in hardware. A parallel algorithm, the parallel Pollard's Rho, is described in [25]. To our knowledge, no actual results of a hardware implementation of Pollard's Rho algorithm for solving the ECDLP have been published yet.

With the paper at hand, we will present first results of a hardware implementation of an attack against ECC over prime fields, giving raise to a security evaluation of ECC taking cryptanalytical hardware into account. We propose an efficient architecture for the parallel variant of Pollard's Rho method and its implementation in hardware. Besides the hardware architecture which has been completely programmed in VHDL and realized on a contemporary low-cost FPGA (Xilinx Spartan-3), this project involves external software components required for managing the hardware components. Based upon first results of the running hardware implementation, we estimate the expected runtime to break ECC for actual security parameters. In this context, we give estimates for an ASIC design, solely dedicated to tackle actual ECC Challenges [3] with our architecture.

We will commence with a brief review of related publications in Section 2, including work regarding ECC operations in hardware as well as efficient attacks on ECC. Section 3 covers relevant mathematical background on cryptanalysis of ECC. Since our focus is a security evaluation of a generic ECC over prime fields, we will not describe attacks on particular curves known as 'weak' curves. Furthermore, we will not restrict ourselves to particular standardized curves, allowing for an efficient implementation. Section 4 provides the choice of parameters for the Multiple Processor Parallel Rho (MPPR) algorithm and the construction of an architecture of MPPR with an emphasis on its application in hardware. The realization of the presented MPPR architecture on FPGAs is discussed in Section 5. We will analyze our results in Section 6 and compare the runtime and resource consumption of our software and hardware implementation. With the results, we are able to give an estimate of the expected runtime of attacks against ECC over $GF(p)$ and recent ECC challenges. Finally, Section 7 summarizes most important findings and concludes this contribution.

2 Previous Work

We briefly summarize previously published results of importance to this contribution. Relevant work includes publications describing attacks against ECC in hardware. To our knowledge, there is no actual hardware implementation realizing such an attack. Since the method of choice for attacking ECC is Pollard's Rho which involves EC group operations, recent results on the efficiency of EC group operations over $GF(p)$ in hardware are of major interest.

2.1 Hardware for ECC over Prime Fields $GF(p)$

The work of [20] suggests a design for a scalable $GF(p)$ elliptic curve processor in programmable hardware. The authors implemented an arithmetic unit on an FPGA capable to perform field additions and semi-systolic multiplications required for projective EC computations. The system runs at 40 MHz and requires 11,416 Look-Up Tables (LUTs), 5,735 Flip-Flops, and 35 Block-RAMS for computations on the EC over $GF(2^{192}-2^{64}-1)$. A point operation can be performed in 2,976 clock cycles (62 μs).

In [21], the authors follow a similar approach: The architecture is based on a five-layered design using Montgomery representation for fast modular reduction. Computations are preferably done in projective coordinates but an interface for using affine points as input is offered. For curves defined over $GF(p)$ with p of size of 160 bit, a point addition and doubling can be computed in 6776 (74 μs) and 6438 clock cycles (70 μs), respectively. The architecture can be operated at a maximum clock speed of 91 MHz on a Virtex-E XCV1000 FPGA. The area consumption is equivalent to [20].

The authors of [4] proposed an architecture relying only on affine coordinates. Compared to hardware for projective coordinates, the additional inverter circuit reduces the overall runtime for a single point computation but yields to an increase in area. According to the authors, an area consumption of 1,854 slices and a maximum clock speed of 40 MHz can be achieved on a Xilinx Virtex-2 XC2V2000 FPGA for a bit length of p of 160 bit. A point addition and doubling requires 809 (20.2 μs) and 972 clock cycles (24.3 μs), respectively.

2.2 Hardware Attacks against ECC

Proposals of hardware-based attacks are very rare and those which exist do not apply to the scope of this work. The most important work in this field is provided by [25]: Besides an algorithmic improvement to allow for efficient parallelization of Pollard's Rho method, the authors estimate the cost of a dedicated hardware solving the DLP over a curve over $GF(2^{155})$ to 32 days for US$ 10 million. However, specific details of the parametrization of the algorithm are omitted. In contrast to curves over $GF(2^m)$, curves over GF(p) have not been examined yet.

3 Mathematical Background

In the following, we will briefly introduce to the mathematical background relevant for this paper. We will start with a review of the Elliptic Curve Discrete Logarithm Problem (ECDLP) and describe possible ways of solving it. For the scope of this paper, a parallel variant of Pollard's Rho method is of major interest and will be described in more detail.

3.1 The Elliptic Curve Discrete Logarithm Problem

The ECDLP basically is an extension of the DLP in finite fields. The security of many cryptographic protocols such as, e.g., the Diffie-Hellman key exchange [7] and the ElGamal encryption scheme [8] is based on the DLP.

ECDLP over $GF(p)$

Let p be a prime with $p > 3$ and $\mathbb{F}_p = GF(p)$ the Galois Field over p. Given the elliptic curve

$$E : y^2 = x^3 + ax + b,$$

with $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$, two points $P, Q \in E(\mathbb{F}_p)$, and $\langle P \rangle$ generator of a sufficiently large subgroup. Find the integer ℓ such that

$$\ell \cdot P = Q, \tag{1}$$

where $Q \in \langle P \rangle$ holds. The parameter ℓ often is denoted as the elliptic curve discrete logarithm $\ell = \log_P(Q)$.

The definition of the DLP in finite fields applies modular multiplication as group operation. Exponentiation denotes a repeated group operation. In the case of elliptic curves, the modular multiplication transforms to the so-called point addition. The exponentiation translates to a repeated point addition, often denoted as point multiplication, forming the basis of the ECDLP. Index-calculus methods against the DLP over finite fields are based on multiplicative characteristics of such and are not applicable to cyclic groups of points over elliptic curves. Therefore, currently only square-root algorithms remain as candidates for solving the ECDLP and will be discussed in the next section.

3.2 Known Algorithms to solve the ECDLP

Most known attacks on ECC have exponential complexity. This statement holds for generic curves and excludes attacks on special subclasses like supersingular and anomalous curves. This paper intends to analyze the security of cryptosystems based on cryptographically strong curves and, thus, weak curves are not considered.

The ECDLP, i.e., the solution ℓ of Equation (1) can be computed by using following techniques:

- Naïve exhaustive search: This method sequentially adds the point $P \in E(\mathbb{F}_p)$ to itself. The addition chain $P, 2P, 3P, 4P, \ldots$ will eventually reach Q and yield ℓ with $\ell \cdot P = Q$. In the worst case, this computation can take up to $n - 1$ steps where $n = ord(P)$, making this attack infeasible for practice when n is large.

- Baby Step Giant Step (BSGS): The BSGS algorithm is an improvement to the naïve exhaustive search [24]. For $n = ord(P)$, temporary memory for about $\sqrt{n}$ points and approximately an additional $\sqrt{n}$ computational steps are required. Due to its high memory complexity, BSGS is not of interest within this contribution.

- Pollard's Rho: The Pollard Rho attack was proposed by J. Pollard in 1978 [22] and is a collision based algorithm relying on a random walk in a cyclic group and, thus, can be applied to the group of points generated by P on the elliptic curve. The random walk computes a trail of points on the elliptic curve and eventually ends in a cycle, revealing the

solution of the ECDLP. Although having a similar time complexity of $\sqrt{\pi n/2}$ compared to BSGS, Pollard's Rho is superior due to its negligible memory requirements. In combination with adequate parallelization, Pollard's Rho method is the fastest known attack against ECC [12]. In the following, we differentiate between the original Pollard's Rho method, the Single-Processor Pollard's Rho (SPPR), and its parallelized variant, the Multiple-Processor Pollard's Rho (MPPR) which is discussed in more detail in the following.

3.3 Multi-Processor Pollard-Rho (MPPR)

The MPPR for solving the ECDLP has been proposed in [25] and basically is a variant of the previously presented SPPR with some modification for optimal parallelization, yielding a linear speed-up with the number of available processors. Because of the fact that a set of processors $\mathcal{W}$ can not easily contribute to the work on a single trail due to limitations in data distribution, another approach is favored for the MPPR.

Each processor $w_i \in \mathcal{W}$ starts an individual trail but does not focus on terminating in a cycle like in SPPR. The primary goal is to find collisions of different computationally independent trails. For this purpose, a selection criteria for points on the trail is defined, marking a small partition of all computable points as "distinguished". In our case we simply assign this property to points with the bits of an x-coordinate showing a specific number of consecutive zeros. When a trail arrives at a so-called Distinguished Point (DP), the corresponding processor transmits the DP to a central server which keeps track of all submitted DP and checks for duplicates. In case of a duplicate, a collision is found and the algorithm terminates successfully.

The difference between SPPR and MPPR becomes apparent from Figure 1. Instead of a single trail, every instance of Pollard's Rho computes its own trail. Every dark spot in the trail of a processor corresponds to a DP. An eventual collision of two trails in a common distinguished point is highlighted in the figure for the processors w_3 and w_4.

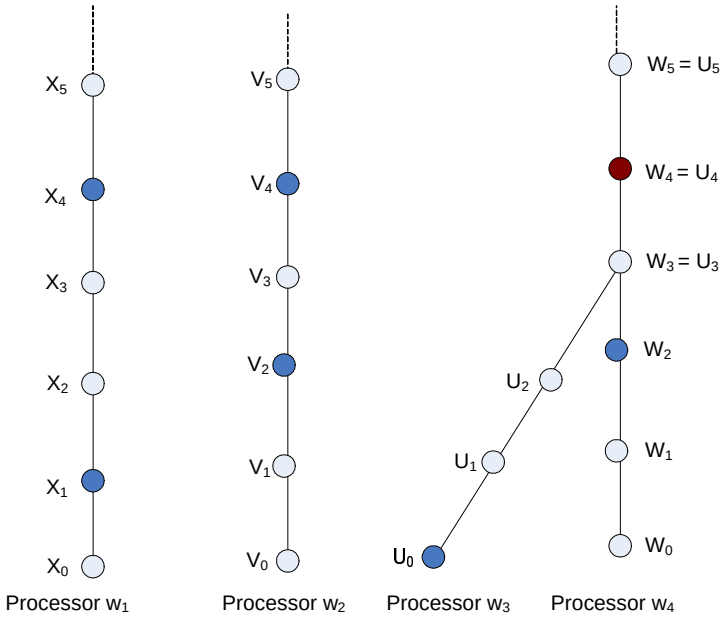

Figure 1: Multi-Processor Pollard's Rho

In contrast to the application of completely independent SPPR instances, the centralized collection of DP in the MPPR achieves a linear speed-up with multiple instances. On average, a collision occurs after the computation of $T = \sqrt{\pi n/2} + c$ points, where n is the order of the actual subgroup. The central unit will detect the merging of two trails when the next DP in the (joint) trail is found. The term c takes the additional overhead of collecting points in a joint trail into account. If we assume that all available processors directly contribute to the threshold T, the workload of a single processor $w_i \in \mathcal{W}$ is given by Equation (2) [25]. Algorithm 1 depicts the MPPR method.

$$\sqrt{\pi n/2}/\left|\mathcal{W}\right| + c \tag{2}$$

Algorithm 1 Multi-Processor Pollard's Rho [12]

Input: $P \in E(\mathbb{F}_p); n = ord(P), Q \in \langle P \rangle$
Output: The discrete logarithm $\ell = log_P(Q)$

Select the size s of a finite set with random points
Select a partitioning function $g : \langle P \rangle \rightarrow \{0, 2, \ldots, s-1\}$
Select a set D out of $\langle P \rangle$ satisfying the distinguished point property
for $i = 0$ to $s - 1$ **do**
  Select random coefficients $a_i, b_i \in_R [1, \ldots, n-1]$
  Compute i-th random point $R_i \leftarrow a_i P + b_i Q$
end for
for each parallel processor **do**
  Select starting coefficients randomly $c, d \in_R [1, \ldots, n-1]$
  Compute a starting point $X \leftarrow cP + dQ$
  repeat
    if $X \in D$ is a distinguished point **then**
      Send (c, d, X) to the central server
    end if
    Compute partition of current point $i = g(X)$.
    Compute next point $X \leftarrow X + R_i$; $c \leftarrow c + a_i \; mod \; n$; $d \leftarrow d + b_i \; mod \; n$
  until a collision in two points was detected on the server
end for
Let the two colliding triples in point Y be (c_1, d_1, Y) and (c_2, d_2, Y)
if $c_1 = c_2$ **then**
  return failure
else
  Compute $\ell \leftarrow (c_1 - c_2)(d_2 - d_1)^{-1} \; mod \; n$
  return ℓ
end if

4 An Efficient Hardware Architecture for MPPR

For the implementation of Algorithm 1, we need a star topology in which a central server is collecting distinguished points from a multitude of attached computational processors $w_i \in \mathcal{W}$. For the realization of MPPR the server requires following features:

- A **communication controller** for data exchange with the $|\mathcal{W}|$ point processors.

- A **database** for storing the tuples (c, d, Y) for a point $Y = cP + dQ$ in a sorted table according to Y for efficient point recovery.
- A unit for **validating distinguished points** received from a processor w_i. This step is mandatory in terms of defective processors which might spoil the entire computational result by eventually corrupted results from a point processor w_i.
- An arithmetic unit for **computing the discrete logarithm** from a detected collision.
- An **EC generator** for testing purposes. This unit is optional when external curve parameters are specified.

In contrast to the connected point processors which are designed to run on special-purpose hardware, the central server has only little requirements regarding computational power and throughput. Thus, it is sufficient to model our central station in software which simplifies the development process. Details of the parametrization of MPPR for hardware can be found in [11]. Figure 2 depicts the data flow of an MPPR system between the contributing parties.

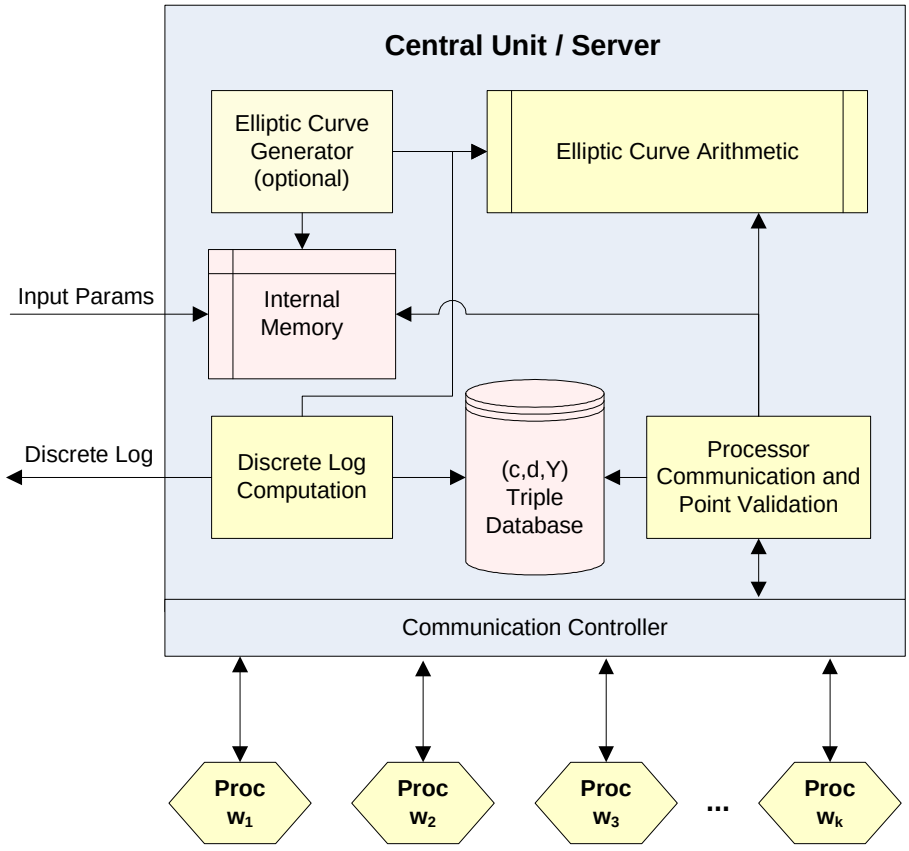

Figure 2: A component model for MPPR

4.1 Proposed Architecture

With the central server implemented in software, we can focus on the point processors $w_i \in \mathcal{W}$ which are subject to the hardware implementation. In this work, we will describe an FPGA implementation of the point processors. Due to the relatively low cost at low quantities and the

reconfigurability, the choice of FPGAs seems optimal for a first evaluation of MPPR in hardware. Remark that a possible ASIC implementation would dramatically decrease the monetary effort of MPPR if produced at high volumes. However, for solving ECDLP over smaller groups ($\leq$130 bit), an FPGA-based design can be implemented with COPACOBANA which is a massively parallel computer of 120 FPGAs of type Xilinx SPARTAN-3 XC3S1000 [16]. Based on the given cost and computational power of COPACOBANA, we are able to estimate the expense for an FPGA-based MPPR attack fairly accurate.

For the implementation of a point processor, the computation of a trail of consecutive points consists of four steps:

1. **DP Detection.** We need to detect if the current point X is distinguished. In hardware, we can agree on a simple DP property which is satisfied when the topmost δ bits of the x-coordinate of the current point $X = (x, y)$ are zero. This can be easily implemented by a δ-bit comparison. In case we found such a point, we will transmit it to the central server.

   Note that an efficient verification of the distinguished property does only work in affine coordinates. Affine coordinates suffer from expensive field inversions which are mandatory to compute the group addition. Projective coordinate systems avoid inversions by encoding the inversion in a separate variable at the cost of additional field multiplications. Unfortunately, the projective point representation is not unique and does not allow for an efficient check of the distinguished property. The reason for this is that an affine point $P = (x, y)$ has $p - 1$ different projective representations, e.g., $P = (X, Y, Z)$ satisfying $x = X/Z$ and $y = Y/Z$. The computational effort to detect a DP from its projective representation usually is more complex than the use of affine coordinates along with a very efficient DP detection. Consequently, we will use affine coordinates in our implementation.

2. **Partitioning.** In order to select the next random point R_i and its corresponding coefficients a_i and b_i, we need to identify a partition $i = 0, \ldots, s - 1$ according to the current point X. In hardware, it is straightforward to choose a power of 2 for s since we simply can use the $\log_2 s$ least significant bits from x to map to the partitioning value i. We choose a (heuristically determined) optimal value $s = 16$ (see [11] for details).

3. **Point Operation.** We need to update the current point by adding a random R_i which requires a point addition or, in case of $X = R_i$, a point doubling. The latter case actually means that a collision is not detected among two distinguished points rather than the current point X and a random point R_i. Due to the fact that this case is very unlikely with a probability of $Pr(X = R_i) = s/n$, we will omit the additionally required logic for this situation.

4. **Coefficient Update.** Finally, we need to update the corresponding coefficients c and d for our updated point X with $X = cP \cdot dQ$. To do so, we add the random coefficients $c = c + a_i \bmod n$ and $d = d + b_i \bmod n$ according to the selected partition i.

In the following, we introduce the design of a top layer in Subsection 4.1.1 which cares for administrative tasks such as, e.g., providing a centralized distinguished point buffer (DPB) as well as a processor controller (PC) which is responsible for the control of the several computational cores. In Subsections 4.1.2 and 4.1.3 we will describe a single computational core and its arithmetic unit, respectively.

4.1.1 Top Layer

The top layer is composed of the previously mentioned DPB, PC, and a physical Communication Controller (CC), as depicted by a schematic outline in Figure 3. The CC is required to control the data input and output from and to the central server, respectively. For the implementation at hand, the CC was realized with a serial line controller (UART). In the next stage (with COPACOBANA), we will use a simple bus-driven controller, providing a higher data rate. Note that no high data communication between FPGA and central server is required. However, if the number of PR cores increases we still can adjust the distinguished property in order to decrease the point throughput by longer computational trails, reducing the load of the communication path.

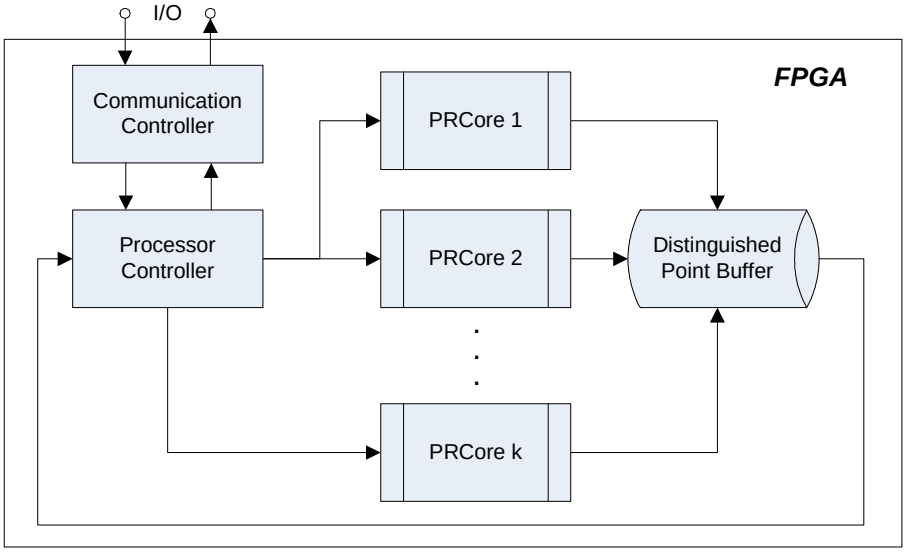

Figure 3: Top layer of an FPGA-based point processor

A gain in performance can be achieved by using more computational cores on a single FPGA, computing many trails in parallel. Obviously, this procedure requires a careful management regarding the data input and output. With many PR cores and the centralized buffer, the access to the DPB demands multiplexing and buffer locking capabilities to prevent two or more cores from writing to the buffer at the same time. On the other hand, the DPB is very useful since it allows for an independent external access to the stored distinguished points without the need to interrupt a core from computing. Figure 3 shows the computational cores as an entity which will be discussed in detail in the following.

4.1.2 Core Layer

For an optimal design, all operations of the core should make use of extensive resource sharing, e.g., by combining all costly field operations in a central Arithmetic Unit (AU) for $GF(p)$ operations. The AU entity will be discussed in Subsection 4.1.3.

An important component of the core layer is the core controller. It is primarily responsible for managing the operations of the AU and delegating its output to memory locations. Another constituent of this layer is the main memory, providing the operands to the AU. A schematic overview of the core layer architecture is given by Figure 4.

The AU demands for three concurrent inputs: Two inputs provide the operands $IN1$ and $IN2$, one input provides the modulus MOD. The modulus can take two possible values: $n = ord(P)$ for updating the coefficients and p for all other computations. The operands change with every single computation. For the operands, we use a dual-port block memory module which is available on the actual FPGA-type. The dual-port memory contains all k-bit variables for the actual point addition including the starting point X, associated coefficients c, d, and temporary values. Besides, it can hold the random point data $(R_i = (x_i, y_i), a_i, b_i)$ for $i = 0...s-1$ in a separate region. Hence, variables and random data is available via a common access path requiring no further multiplexing. For the modulus MOD we use a separate single-port memory module. Compared to two individual registers with multiplexers and separate control lines, this approach is more area efficient. Details of the optimization of the point processor are discussed in [11].

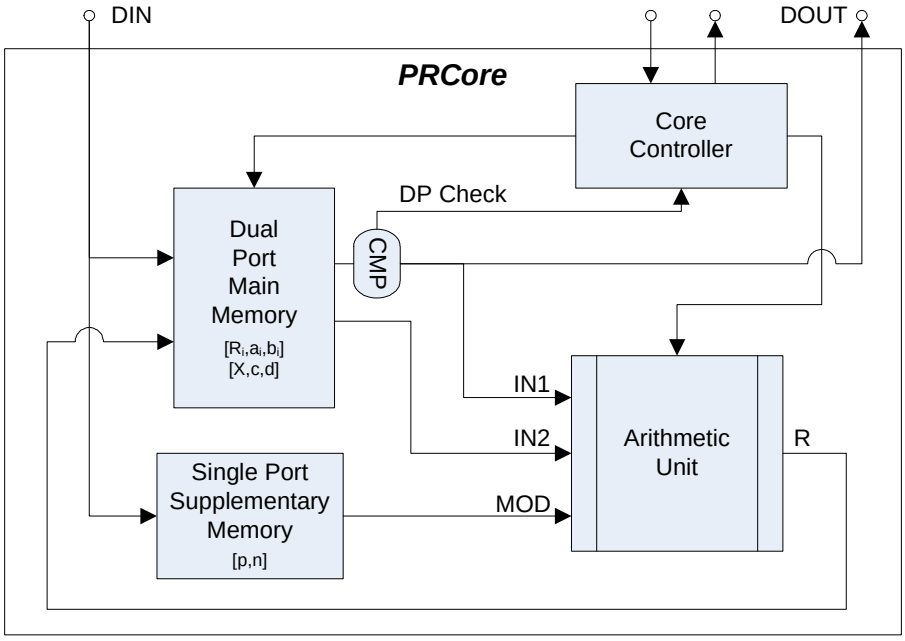

Figure 4: Core layer of an FPGA-based point processor

4.1.3 Arithmetic Unit

The overall runtime of an implementation of MPPR is dominated by the performance of the field arithmetic, used for the point addition and coefficient updates. Hence, the efficient implementation of the arithmetic functions such as field addition, subtraction, multiplication, and inversion is crucial for the performance of the overall system. Due to the predominance of the modular inversion within the point addition function, the optimization of other operations has not been considered since a dramatic reduction of the critical path is not expected.

The arithmetic unit provides the functionality of field addition, subtraction, multiplication, and inversion required by the upper layers. Modular addition and subtraction are rather simple operations. In contrast, modular multiplication and inversion takes much more computational effort. For an efficient implementation of such we use coordinates in Montgomery domain, allowing for algorithms which replace a costly modular reduction by divisions by two (implemented by right shifts by one) [18]. Thus, we use a Montgomery multiplication and a modified Kaliski

inversion algorithm [5]. All field operations require k-bit additions and subtractions which we realize with Xilinx IP cores. The IP cores are precompiled and highly optimized for a specific FPGA and provide a better performance-area ratio than conventional implementations.

Excluding overhead for the control logic, the field addition and subtraction can be implemented with one k-bit adder/subtracter IP core, one k-bit register, and three k-bit multiplexers. A very space-efficient implementation of the Montgomery multiplication can be realized with one $k+1$-bit adder/subtracter with carry-out, two $k+1$-bit shift-registers, and a single $k+1$-bit multiplexer. The Kaliski inverter however, is the most area-consuming part: It requires three $k+1$-bit adder/subtracters, four $k+1$-bit shift-registers, and six $k+1$-bit multiplexers. For detailed specifications of the AU please refer to [11].

The final design of our AU is shown in Figure 5. For the sake of clarity, we combined the logic for the shifting units and the multiplexing units in corresponding blocks.

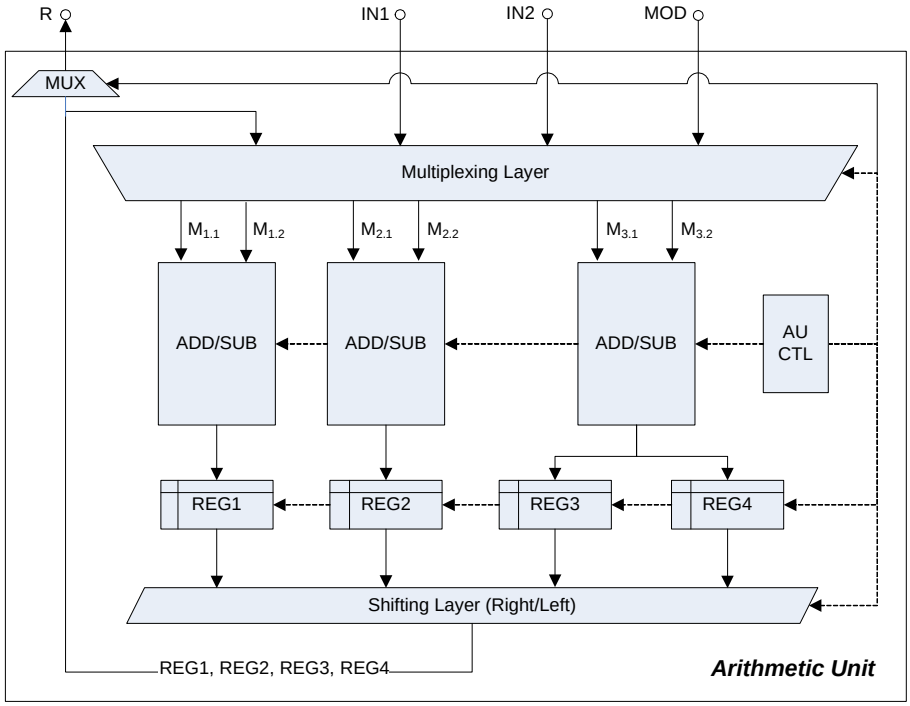

Figure 5: Arithmetic layer of an FPGA-based point processor

5 Implementation

The proposed architecture has been implemented on a Xilinx Spartan-3 device (XC3S1000) and corresponding results are presented in Subsection 5.1. For comparison and debugging purposes, a software implementation of the complete MPPR has been realized on a conventional processor (Pentium M). Furthermore, we estimate the cost and performance of an ASIC implementation of the MPPR based on the running FPGA implementation in Subsection 5.3 and finally compare our findings in Subsection 5.4.

5.1 Synthesis Results

We synthesized the design for various bit sizes k and the maximum number of cores fitting on one FPGA. Table 1 shows the area and clocking constraints for a Xilinx XC3S1000, providing 17,280 logic cells in 7,680 slices with a variable number of cores. Note, that the number of occupied slices includes all top-layer components for management and communication, as well.

k	**Cores**	**Min. Period**	**Max. Clock**	**Slices**	**Usage**	**Slices/Core**
160	2	25.0 ns	40.0 MHz	6,450	83 %	3,230
128	3	24.9 ns	40.1 MHz	7,561	98 %	2,520
96	3	20.8 ns	48.1 MHz	6,091	79 %	2,030
96	4	22.6 ns	44.3 MHz	7,564	98 %	1,890
80	4	19.6 ns	50.9 MHz	6,820	88 %	1,710
64	4	18.2 ns	54.8 MHz	5,789	75 %	1,450
64	5	19.2 ns	52.0 MHz	6,817	88 %	1,360

Table 1: Area and timing constraints on XC3S1000 FPGA

Apparently, architectures with a device usage close to 100 % have a significantly longer propagation delay than those with some unused FPGA area (e.g., $k = 128$ and $k = 96$). This effect can be directly traced back to the more condensed packaging which leaves only little flexibility for extensive route optimizations after synthesis. Hence, longer signal paths need to be taken into account and lead to a lower clock frequency.

5.2 Throughput

With the complexity of all core functions, we can compute an exact number of cycles required for an iteration of MPPR. Along with the maximum clock frequency, detailed numbers for the throughput of the whole design can be given.

An elliptic curve point addition in affine coordinates can be performed using 6 subtractions, 3 multiplications, and a single inversion. The update of coefficients demands another two additions and a few administrative cycles. Finally, we can provide a list for the overall cycle count of a single MPPR iteration as shown in Table 2, giving raise to following complexity function:

$$T(k) := T_{CMP}(k) + T_{RST} + T_{FIN} = 5k + 55. \tag{3}$$

Operation Type	**Count**	T_{CMP}	T_{RST}	T_{FIN}
Partitioning	1	1/2	-	-
DP Check	1	1/2	-	-
Subtractions	6	6	12	6
Multiplications	3	$3(k+2)$	6	3
Inversion	1	$2(k+2)$	2	1
Addition	2	2	4	2
Total	14	$5(k+2)+9$	24	12

Table 2: Required cycles for one MPPR iteration with bit size k

With Equation (3), we can easily compute the performance for the selected architectures including the number of required cycles per iteration and the number of points per second.

k	**Cores**	$T(k)$	**Time per Pt**	**Pts per Core/sec**	**Total Pts/sec**
160	2	855	21.400 μs	46,800	93,600
128	3	695	17.300 μs	57,800	173,000
96	3	535	11.100 μs	90,000	270,000
96	4	535	12.100 μs	82,700	331,000
80	4	455	8.940 μs	111,900	447,000
64	4	375	6.840 μs	146,200	585,000
64	5	375	7.210 μs	138,600	693,000

Table 3: Performance of MPPR on XC3S1000 FPGA

Remark, that the numbers above exclude additional cycles for writing to the distinguished point buffer. Only when computing the ECDLP in small groups, the buffer might become a bottleneck and the additional cycles for data output have to be taken into account. I.e., a larger number of MPPR cores per FPGA increases the demand for a large output buffer. However, this can easily be avoided by reducing the size of the distinguished point set D. A direct consequence are longer search trails for distinguished points and, thus, collisions of two or more cores requesting a buffer lock at the same time will become extremely unlikely.

5.3 Estimate for an ASIC Design of MPPR

Based on the results from the FPGA implementation, we can estimate the performance and cost of an ASIC realization. For an appropriate estimate, our application specific design will be limited to 10 million gates which is approximately a tenfold of the logic resources of the Xilinx XC3S1000 FPGA and which we believe to be fairly realizable. Furthermore, we can expect to run such a chip at clock rates beyond the scope of low-cost FPGAs. Hence, a maximum clock frequency of 500 MHz seems realistic with the chosen algorithm. Table 4 provides an estimate of the throughput for a possible chip design for MPPR over $GF(p)$ with a k-bit prime p and a maximum possible number of MPPR cores per chip.

Please note that due to limited time we have not yet synthesized the VHDL code for an actual ASIC library. Hence, all given numbers in this paper are estimates and are subject to change in case of a future ASIC simulation. However, we believe that the estimates are fairly accurate and will not differ in an order of magnitude. Moreover, in case of a deviation we expect an improvement of area/ time complexity and, thus, the numbers provided can be seen as an upper bound on the hardware complexity of MPPR.

k	**# MPPR cores**	**Performance (pts/sec)**
64	50	$6.66 \cdot 10^7$
80	40	$4.40 \cdot 10^7$
96	40	$3.74 \cdot 10^7$
128	30	$2.16 \cdot 10^7$
160	20	$1.17 \cdot 10^7$

Table 4: MPPR performance estimates for an ASIC design (10^7 gates, 500 MHz)

5.4 Projected Runtimes for Solving ECDLP on Different Platforms

Along with the given throughput from Subsection 5.2, projections are performed for bit lengths $k = \{64, 80, 96, 128, 160\}$ using a distinguished point proportion of $\Theta = 2^{-24}$. We will regard following architectures: First, projections are done for the software reference, running on a conventional off-the-shelf processor (Pentium M 735@1.7 GHz). Secondly, we will estimate the performance on the XC3S1000 FPGA. Thirdly, an estimate for dedicated ASIC will be given for bit sizes of 128 bit and above[1]. Clearly, the performance of the hardware architecture heavily depends on the varying bit size, indicated by different runtimes and varying area consumption. For small bit lengths, several cores can be realized on the FPGA and, thus, slightly decrease the maximum clock speed of the entire system. However, the additional computational power compensates the lower clock speed.

We compute the expected number of points for a specified bit length to determine the ECDLP of an associated curve. At this time, we discard any negative effect of overwriting points on the central station which in fact needs to be considered when the number of available distinguished points becomes too huge to be stored without compression. Table 5 shows the upper (T_H) and lower (T_L) bounds on the number of computed points and the corresponding number of expected DP to be stored on the server, S_H and S_L, respectively. For details, please refer to [11].

k	$T_L = \sqrt{\pi 2^{k-1}}/2 + c$	$T_H = \sqrt{\pi 2^k}/2 + c$	$S_L = \Theta \cdot T_L$	$S_H = \Theta \cdot T_H$
64	$2.71 \cdot 10^9$	$3.82 \cdot 10^9$	$1.61 \cdot 10^2$	$2.28 \cdot 10^2$
80	$6.89 \cdot 10^{11}$	$9.74 \cdot 10^{11}$	$4.11 \cdot 10^4$	$5.81 \cdot 10^4$
96	$1.76 \cdot 10^{14}$	$2.50 \cdot 10^{14}$	$1.05 \cdot 10^7$	$1.49 \cdot 10^7$
128	$1.16 \cdot 10^{19}$	$1.64 \cdot 10^{19}$	$6.89 \cdot 10^{11}$	$9.74 \cdot 10^{11}$
160	$7.58 \cdot 10^{23}$	$1.07 \cdot 10^{24}$	$4.52 \cdot 10^{16}$	$6.39 \cdot 10^{16}$

Table 5: Estimated MPPR runtime and number of stored distinguished points

If we assume a geometrical distribution of distinguished points we can show that only negligible effects from overwriting points are expected for bit sizes ≤ 96 bit. This statement is true for an available size of server storage of $2^{24} \approx 1.68 \cdot 10^7$ points.

Table 6 presents the results for the expected computation time with a single chip (Pentium M, XC3S1000 FPGA, and ASIC). Obviously, a successful computation of the ECDLP for $k > 96$ seems to be unrealistic with any of the presented architectures when considering only a single chip.

k	$Mean(T_H, T_L)$	**Pentium M**	**XC3S1000**	**ASIC**
64	$3.25 \cdot 10^9$	5.14 h	78.1 min	-
80	$8.32 \cdot 10^{11}$	70.5 d	21.5 d	-
96	$2.13 \cdot 10^{14}$	55.1 y	20.4 y	-
128	$1.40 \cdot 10^{19}$	$4.86 \cdot 10^6$ y	$2.55 \cdot 10^6$ y	$2.05 \cdot 10^4$ y
160	$9.15 \cdot 10^{23}$	$3.78 \cdot 10^{11}$ y	$3.10 \cdot 10^{11}$ y	$2.48 \cdot 10^9$ y

Table 6: Expected runtime for MPPR on a single chip

[1]For lower bit sizes, the relatively high NRE costs outweigh the production costs.

6 Security Evaluation of ECC

For a detailed security evaluation, we regard attacks against ECC based on the three previously presented architectures: The software implementation on a cluster of conventional off-the-shelf processors, an FPGA cluster built of the FPGA-type of our implementation, and a dedicated ASIC hardware consisting of a cluster of the ASICs presented in Subsection 5.3.

We will estimate the cost of such cryptanalytical systems for different scenarios: We consider a successful attack against ECC in one year and compare the results to the previously proposed hardware attacks against RSA in [23, 9]. Finally, we estimate the approximate cost to solve the ECC challenges given by [3].

6.1 Considered Platforms and Related Costs

A direct line-up of the software implementation against the optimized hardware architectures would be not very meaningful. The reason being is, first, that we did not optimize the software reference with the same effort than we did in the case of the hardware implementation. Since the optimization of the software is not within the scope of this contribution, we will take possible improvements by using a correction factor of 1.25 into account. I.e., a possible speed-up by 25% can be achieved by using assembler code, placing our software solution into a more competitive position against its relative in hardware. Secondly, we have to take the different monetary cost of general purpose processors, FPGAs, or ASICs into account. For the sake of simplicity, we do not consider power consumption[2]. Thirdly, the processor (Intel Pentium M) of the software implementation possesses approx. 140 million transistors, neglecting possible additional memory besides the processor's cache. For comparison, the XC3S1000 FPGA has only one million system gates. To overcome this inherent imparity and to provide a fair comparison of the platforms, we will use a very simple metric which is of high practical relevance: The *cost-performance ratio*. We can relate the software and hardware performance with respect to the throughput in point computations and the corresponding monetary cost. The monetary aspect is of central importance when investigating the feasibility of attacks of actual ECC.

We assume the cost of a Pentium M 735 processor and a Xilinx XC3S1000 FPGA to be approximately US$ 220 and US$ 50 per chip, respectively[3]. Additionally, we need housing and peripheral interconnections for each chip. In the case of FPGAs, the realization of a cost-efficient parallel architecture called COPACABANA is presented in [16]: The architecture manages up to 120 low-cost FPGAs (Xilinx XC3S1000) at a total cost of approximately US$ $10,000$ including material and production costs. Taking the amount of US$ $10,000$ as a reference, we can estimate a corresponding number of workstations with Pentium M processors. Assuming additional US$ 180 for housing, mainboard, main memory, and physical storage (such as a hard disk drive or a flash memory device for the operation system), we estimate the cost for a single workstation to be US$ 400. Thus, a cluster of 25 Pentium M workstations costs approximately the same than the COPACOBANA machine with 120 low-cost FPGAs. For the ASIC design from Subsection 5.3, we assume a cost of US$ 50 per chip including overhead, which seems to be a fair assumption for the production of large quantities. Hence, for US$ $10,000$ we can build an ASIC cluster consisting of 200 ASICs. Due to the relatively high NRE costs of ASICs however, such a design will only be considered when targeting ECC with bit sizes of 128 bit and

[2]However, we believe the power consumption of special-purpose hardware to be far below that of general purpose architectures.

[3]Market prices for low quantities; Prices at high volumes might differ.

above and many chips have to be produced. For the sake of simplicity, we do not take further aspects such as power consumption into account. Table 7 compares the computational power for a US$ 10,000 MPPR-attack in terms of software and hardware implementations.

k	**Pentium M cluster**	**FPGA cluster**(COPACOBANA)	**ASIC cluster**
160	$1.92 \cdot 10^6$ pts/sec	$11.2 \cdot 10^6$ pts/sec	$2.34 \cdot 10^9$ pts/sec
128	$2.28 \cdot 10^6$ pts/sec	$20.8 \cdot 10^6$ pts/sec	$4.32 \cdot 10^9$ pts/sec
96	$3.07 \cdot 10^6$ pts/sec	$39.7 \cdot 10^6$ pts/sec	$7.48 \cdot 10^9$ pts/sec
80	$3.41 \cdot 10^6$ pts/sec	$53.7 \cdot 10^6$ pts/sec	$8.79 \cdot 10^9$ pts/sec
64	$4.39 \cdot 10^6$ pts/sec	$83.2 \cdot 10^6$ pts/sec	$13.3 \cdot 10^9$ pts/sec

Table 7: MPPR performance comparison for a US$ 10,000 investment

6.2 Breaking ECC in One Year

The number of required days of computation can only be reduced by parallelization, i.e., a massive employment of many instances. If we define a specific time constraint to be met, we can compute the number of required chips. Following the attempt to break ECC with a bit length of k in a maximum time of one year (365 days) will lead us to the number of requried chips represented by Table 8.

k	**# Pentium M**	**# XC3S1000**	**# ASIC**
80	1	1	-
96	56	21	-
128	$4.86 \cdot 10^6$	$2.55 \cdot 10^6$	$2.05 \cdot 10^4$
160	$3.78 \cdot 10^{11}$	$3.1 \cdot 10^{11}$	$2.48 \cdot 10^9$

Table 8: Number of required chips for a single year attack

It will take at least about 378 billion general purpose processors to successfully complete an attack on a $k = 160$ bit ECC within one year. With special-purpose hardware implementing our architecture, 2.6 billion machines of type COPACOBANA, equivalent to the computational power of 310 billion FPGAs, are required. With the ASIC design, 2.5 billion chips are necessary to complete the task. Obviously, the construction of such a giant device is infeasible from a today's perspective.

In the subsequent paragraphs we will compare the findings to the expected security of RSA and discuss the feasibility of solving different ECC challenges with the presented architectures.

6.3 ECC vs. RSA: A Security Comparison

According to the very optimistic estimate in [23], the hardware for breaking a single RSA-1024 modulus within one year will cost approx. US$ 10 million. A different and more conservative security consideration of RSA has been published in [9]: The authors assumed the cost of a successful attack within one year to approximately US$ 200 million. According to the wide-spread opinion, RSA-1024 is considered to provide a similar level of security than ECC-163[4].

[4]ECC-163 is of particular interest, since 163-bit curves are standardized.

The expenses for a successful attack against ECC-163 within one year are equal to the cost of 7.6 billion COPACOBANA machines and amount to approximately US$ $7.6 \cdot 10^{13}$. Hence, attacks of such dimension are far beyond the feasibility of today's (monetary) capabilities.

Due to the vast amount of computations, ECC with bit length of 128 bit and above can only be efficiently attacked with dedicated ASICs. With the estimate from Subsection 5.3, a successful attack against ECC-163 within one year based on $1.16 \cdot 10^{10}$ ASICs will cost approximately US$ $5.8 \cdot 10^{11}$. This amount still is a factor of $58,000$ and $2,900$ higher compared to the estimates for breaking RSA-1024 as described in [23] and [9], respectively. Although all figures are reasonable estimates for recent hardware architectures with no claim for representing a definite lower bound, we can still assume EC-163 to require significantly more efforts than for a successful attack at RSA-1024 when considering our findings.

6.4 ECC Challenges

In 1997, the company Certicom has issued the so-called *ECC challenges* to demonstrate the security of ECC by announcing a list of elliptic curves and associated ECDLP parameters [3]. For numerous bit lengths, a reward for solving such an ECDLP is announced. For ECC over prime fields $GF(p)$, challenges have been defined for the bit sizes

$$k = \lceil \log_2 p \rceil = \{79, 89, 97, 109, 131, 163, 191, 239\} .$$

Certicom provided estimates for the required number of machine days for solving each challenge. However, the runtime estimates are based on a quite outdated Intel Pentium 100 processor. Consequently, we will additionally provide runtimes for our software reference for comparison with the hardware implementations. Table 9 depicts how fast our architectures can solve these challenges.

k	**Certicom Est.** [3]	**Pentium M**	**XC3S1000**	**ASIC**
79	146 d	49.0 d	15.3 d	-
89	12.0 y	4.64 y	1.62 y	-
97	197 y	74.7 y	30.7 y	-
109	$2.47 \cdot 10^4$ y	$5.57 \cdot 10^3$ y	$2.91 \cdot 10^3$ y	-
131	$6.30 \cdot 10^7$ y	$1.40 \cdot 10^7$ y	$7.40 \cdot 10^6$ y	$9.34 \cdot 10^4$ y
163	$6.30 \cdot 10^{12}$ y	$1.09 \cdot 10^{12}$ y	$9.15 \cdot 10^{11}$ y	$1.16 \cdot 10^{10}$ y
191	$1.32 \cdot 10^{17}$ y	$2.17 \cdot 10^{16}$ y	$1.89 \cdot 10^{16}$ y	$2.39 \cdot 10^{14}$ y
239	$3.84 \cdot 10^{24}$ y	$4.44 \cdot 10^{23}$ y	$8.62 \cdot 10^{23}$ y	$1.01 \cdot 10^{22}$ y

Table 9: Expected runtime on different platforms and for different Certicom ECC challenges

Considering the latest unsolved Certicom challenge over $GF(p)$ ($k = 131$ bit), we estimate the required computational power to be at least $62,000$ COPACOBANA machines ($7.40 \cdot 10^6$ FPGAs) for solving the ECDLP within a year. Unlike $k = 160$, this is also an enormous but not an absolutely unrealistic amount of computational power. With $93,400$ ASICs, this challenge would take one year to finish at a cost of approximately US$ $5,000,000$, excluding NRE costs. Analyzing the last solved challenge with $k = 109$ bits, we can state that about 300 COPACOBANA machines are already sufficient to solve this ECDLP in only 30 days. This single-month attack on ECC with $k = 109$ bit can be realized with FPGAs amounting to approx. US$ 3 million. For bit lengths exceeding 131 bit however, the only feasible way to solve the ECDLP is the application of dedicated ASIC devices. Assuming the high-performance ASICs from Subsection 5.3, we can estimate the runtime for MPPR attacks with respect to different

financial background. Table 10 depicts the expected runtimes for solving the ECDLP for different bit lengths and monetary efforts.

	Expected runtimes in years for attacks with			
k	US$ 10^5	US$ 10^6	US$ 10^7	US$ 10^8
128	$1.03 \cdot 10^1$	1.03	0.103	0.0103
160	$1.24 \cdot 10^6$	$1.24 \cdot 10^5$	$1.24 \cdot 10^4$	$1.24 \cdot 10^3$
192	$9.64 \cdot 10^{10}$	$9.64 \cdot 10^9$	$9.64 \cdot 10^8$	$9.64 \cdot 10^7$
256	$1.09 \cdot 10^{21}$	$1.09 \cdot 10^{20}$	$1.09 \cdot 10^{19}$	$1.09 \cdot 10^{18}$

Table 10: Cost-performance consideration of MPPR attacks with ASICs (10^7 gates, 500 MHz, NRE costs excluded)

7 Conclusion

The contribution at hand is the first to present results of a hardware-accelerated attack against the elliptic curve cryptosystem. We present an efficient hardware architecture for the parallelized Pollard's Rho algorithm (MPPR) as described in [25] for groups of points on an elliptic curve over prime fields. Besides the hardware implementation, a software implementation was realized on a conventional processor for comparison and for debugging purposes.

The hardware implementation was accomplished on a contemporary low-cost FPGA (Xilinx Spartan-3) and impressively shows the suitability of the MPPR algorithm for hardware. Compared to existing architectures for $GF(p)$ arithmetic units, we can state that our AU implementation requires $19.995\mu s$ per point addition at $k = 160$ bit occupying about $2,660$ slices of our FPGA. Thus, our implementation provides a better performance at a smaller area cost with respect to [20, 21] and a competitive design concerning the architecture of [4] when balancing the differences between the varying FPGA types.

The corresponding parameters for MPPR have been optimized regarding an area-time efficient implementation of the algorithm in hardware. Considering the latest unsolved Certicom ECC challenge over $GF(p)$ ($k = 131$ bit), we estimate the required computational power to be approximately $7.40 \cdot 10^6$ FPGAs for solving the ECDLP within a year. Unlike ECC-163, this is also an enormous but not an absolutely infeasible amount of computational power. In contrast, an equivalent software implementation requires more CPUs (Pentium M) at a much higher price per piece.

For solving the ECDLP on curves of practical interest however, FPGA implementations are still too costly. In order to derive an estimate of the security of actually used ECC, we provide an estimate for an ASIC implementation based on the outcomes of the FPGA implementation at hand. As a result, ECC turns out to be more secure as commonly believed. The expenses of a successful attack against ECC-163 within one year based on $1.16 \cdot 10^{10}$ ASICs will cost approximately US$ $5.8 \cdot 10^{11}$. Compared to recent (conservative) estimates for a special-purpose hardware attacking RSA-1024 in [9], this amount still is a factor of $2,900$ larger.

References

[1] ANSI X9.62-2005. American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, Accredited Standards Committee X9, `http://www.x9.org`, 2005.

[2] ANSI X9.63-2001. Key Agreement and Key Management Using Elliptic Curve-Based Cryptography. Technical report, Accredited Standards Committee X9, `http://www.x9.org`, 2001.

[3] Certicom. Certicom ECC Challenge, 1997. Available at `http://www.certicom.com`.

[4] A. Daly, W. Marnane, T. Kerins, and E. Popovici. An FPGA implementation of a GF(p) ALU for encryption processors. *Elsevier - Microprocessors and Microsystems*, 28(5–6):253–260, 2004.

[5] A. Daly, L. Marnaney, and E. Popovici. Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic. Technical report, University College Cork, Cork, Ireland, 2004.

[6] T. Dierks and C. Allen. *RFC 2246: The TLS Protocol Version 1.0.* Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA, January 1999.

[7] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22:644–654, 1976.

[8] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31:469–472, 1985.

[9] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, and C. Priplataand C. Stahlke. SHARK — A Realizable Special Hardware Sieving Device for Factoring 1024-bit Integers. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *LNCS*, pages 119–130. Springer-Verlag, August 2005.

[10] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0.* Transport Layer Security Working Group INTERNET-DRAFT, November 1996.

[11] T.E. Güneysu. Efficient Hardware Architectures for Solving the Discrete Logarithm Problem on Elliptic Curves. Master's thesis, Horst Görtz Institute, Ruhr University of Bochum, February 2006. Available at `http://www.crypto.rub.de/imperia/md/content/texte/theses/thesis_gueneysu_mppr.pdf`.

[12] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Verlag, New York, 2004.

[13] Institute of Electrical and Electronics Engineers. *IEEE P1363 Standard Specifications for Public Key Cryptography*, 2000.

[14] S. Kent and R. Atkinson. *RFC 2401: Security Architecture for the Internet Protocol.* Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA, November 1998.

[15] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[16] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker, 2006. Technical Report.

[17] A.K. Lenstra and E.R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[18] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, New York, 1996.

[19] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO '85*, volume LNCS 218, pages 417–426, Berlin, Germany, 1986. Springer-Verlag.

[20] G. Orlando and C. Paar. A scalable GF(p) elliptic curve processor architecture for programmable hardware. volume 2162, pages 356–371, 2001.

[21] S.B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of elliptic curve processor over $GF(p)$. pages 433–443, 2003.

[22] J.M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32(143):918–924, July 1978.

[23] A. Shamir and E. Tromer. Factoring Large Numbers with the TWIRL Device. In *Advances in Cryptology — Crypto 2003*, volume 2729 of *LNCS*, pages 1–26. Springer, 2003.

[24] D. Shanks. Class number, a theory of factorization and genera. *Proc. Symp Pure Math.*, 20:415–440, 1971.

[25] P.C. van Oorschot and M.J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.