Pollard Rho on the PlayStation 3

Joppe W. Bos[1], Marcelo E. Kaihara[1], and Peter L. Montgomery[2]

[1] EPFL IC LACAL, CH-1015 Lausanne, Switzerland
{joppe.bos, marcelo.kaihara}@epfl.ch
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
peter.montgomery@microsoft.com

Abstract. This paper describes a high-performance PlayStation 3 (PS3) implementation of the Pollard rho discrete logarithm algorithm on elliptic curves over prime fields. A record has been set using this implementation by solving an elliptic curve discrete logarithm problem (ECDLP) with domain parameters from a currently standardized elliptic curve over a 112-bit prime field. Solving this 112-bit ECDLP instance required 62.6 PS3 years. Arithmetic algorithms have been designed for the PS3 to exploit the SIMD architecture and the rich instruction set of its computational units. Though our implementation is targeted at a specific 112-bit modulus, most of our implementation strategies apply to other large moduli as well.

Keywords: Elliptic curve discrete logarithm, Pollard rho, Cell broadband engine, SIMD arithmetic

1 Introduction

Elliptic curve cryptography (ECC) [20, 23] is becoming increasingly popular since it allows smaller key-sizes [22] to obtain the same level of security as other widely used public-key cryptographic approaches such as RSA [30]. Government and industry have standardized the use of ECC in, for instance, the Digital Signature Standards (DSS) [38] and the Standards for Efficient Cryptography (SEC) [6]. Here, elliptic curves defined over prime fields ranging from 192 to 512 bits, and from 112 to 512 bits are standardized, respectively.

Processor development seems to be moving away from a single-core towards a multi-core design in order to scale performance through parallelism. The Cell broadband engine (Cell), with its unique heterogeneous architecture, is an interesting example. Its single instruction multiple data (SIMD) organization along with its rich instruction set makes it attractive for accelerating cryptographic operations [8, 2, 7, 3] and cryptanalysis [34, 35].

In this article, the security of ECC – using elliptic curves over prime fields – is evaluated using the relatively low-priced and broadly available multi-core Cell architecture, which is the heart of the video game console PlayStation 3 (PS3). For this purpose, high-performance SIMD arithmetic algorithms have been designed to exploit the features of the instruction set of the Cell. These

SIMD algorithms form the basis of our implementation of the Pollard rho [28] algorithm, the fastest known algorithm to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP). Our implementation has been used to set a record by solving an ECDLP with parameters taken from a 112-bit standardized elliptic curve. Solving this problem required 62.6 PS3 years and ran on a PS3 cluster of more than 200 PS3s in the period January - July, 2009. When run continuously, using the latest version of our code, the calculation would have taken 3.5 months.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the Cell broadband engine. Section 3 recalls the Pollard rho discrete logarithm algorithm together with some optimizations. Section 4 presents efficient arithmetic algorithms aimed at the 112-bit elliptic curve designed to exploit the features of the Cell architecture. Section 5 gives implementation details and performance results. Section 6 concludes the paper.

2 Cell Broadband Engine Architecture

The Cell architecture [16], developed by Sony, Toshiba and IBM, has as a main processing unit, a dual-threaded 64-bit Power Processing Element (PPE) which can offload work to the eight Synergistic Processing Elements (SPEs) [11, 36]. The SPEs are the workhorses of the Cell and the main interest in this article. The SPE consists of a Synergistic Processor Unit (SPU) and a Memory Flow Controller (MFC). Each SPU has a register file of 128 entries called vectors, or quad-words, of 128-bit length and to its own 256-kilobyte Local Store (LS) with room for instructions and data. The main memory can be accessed through explicit Direct Memory Access (DMA) requests to the MFC. The SPUs have a 128-bit SIMD organization allowing sixteen 8-bit, eight 16-bit or four 32-bit integer computations in parallel. The SPUs are asymmetric processors, having two pipelines, denoted as even and odd pipelines. This means that two instructions can be dispatched every clock cycle. Most of the arithmetic instructions are executed on the even pipeline and most of the memory instructions are executed on the odd pipeline. It is a challenge to fully utilize both pipelines always at the same time. The SPEs have no hardware branch-prediction. Instead, the programmer (or the compiler) can provide hints to the instruction fetch unit where a branch instruction will most likely jump to.

An additional advantage of the SPEs is the rich instruction set. For instance, among the available instructions all distinct binary operations $f : \{0,1\}^2 \to \{0,1\}$ are present. The SPEs are equipped with a 4-SIMD multiplier which can compute four 16-bit integer multiplications simultaneously per clock cycle. In addition, a multiply-and-add instruction which performs a 16×16-bit unsigned multiplication, and an addition of a 32-bit unsigned operand to the 32-bit product is provided and has the same time cost as a single 16×16-bit multiplication. This instruction requires the 16-bit operands to be placed in the higher positions of the 32-bit word elements of the vectors. Note that carries are not generated for these instructions.

One of the first applications of the Cell was to serve as the main processor for the Sony's PS3 video game console. The Cell contains eight SPEs, and in the PS3 one of them is disabled. One of the remaining SPEs is reserved by Sony's hypervisor (a software layer which is used to virtualize devices and other resources in order to provide a virtual machine environment to operating systems such as Linux OS). All in all, six SPEs can be accessed when the Linux operating system is installed on a PS3.

3 Preliminaries

3.1 Elliptic Curves over $\mathbb{F}_p$

Let $\mathbb{F}_p$ be a finite field of characteristic $p \neq 2, 3$ and $a, b \in \mathbb{F}_p$ satisfy the inequality $4a^3 + 27b^2 \neq 0$. Informally an elliptic curve $E(\mathbb{F}_p)$ is defined as the set of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ which satisfy the affine Weierstrass equation [33]:

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

These points, together with a point at infinity, denoted as $\mathcal{O}$, form an abelian group where the group operation is point addition and the zero point is the point at infinity. Let $P, Q \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. Then $-P = (x_1, -y_1)$. If $P \neq -Q$ then $P + Q = (x_3, y_3)$ where

$$x_3 = \mu^2 - x_1 - x_2, \quad y_3 = \mu(x_1 - x_3) - y_1 \text{ with } \mu = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q \\ \\ \dfrac{3x_1^2 + a}{2y_1} \text{ if } P = Q. \end{cases} \tag{2}$$

3.2 The Pollard Rho Algorithm

Let E be an elliptic curve over $\mathbb{F}_p$, $P \in E(\mathbb{F}_p)$ a point of order n and $Q = lP \in \langle P \rangle$. Here p is prime and $l, n \in \mathbb{Z}$, in practice p and n are known. The most efficient algorithm in the literature to find $l \bmod n$ for generic curves is Pollard's rho algorithm [28]. The underlying idea of this method is to search for two distinct pairs $(c_i, d_i), (c_j, d_j) \in \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ such that

$$c_i P + d_i Q = c_j P + d_j Q.$$

Then, the discrete logarithm of Q to the base P, i.e. $l = \log_P Q$, can be obtained by computing

$$l \equiv (c_i - c_j)(d_j - d_i)^{-1} \bmod n.$$

This calculation might fail if the inverse of $(d_j - d_i)$ does not exist. In practice, n is prime since one first reduces the calculation of the discrete logarithm to the computation of the discrete logarithm in the prime order subgroups of $\langle P \rangle$ [27].

The occurrence of two such distinct pairs is called a *collision*. Given an iteration function $f : \langle P \rangle \to \langle P \rangle$, the Pollard rho method calculates a sequence of

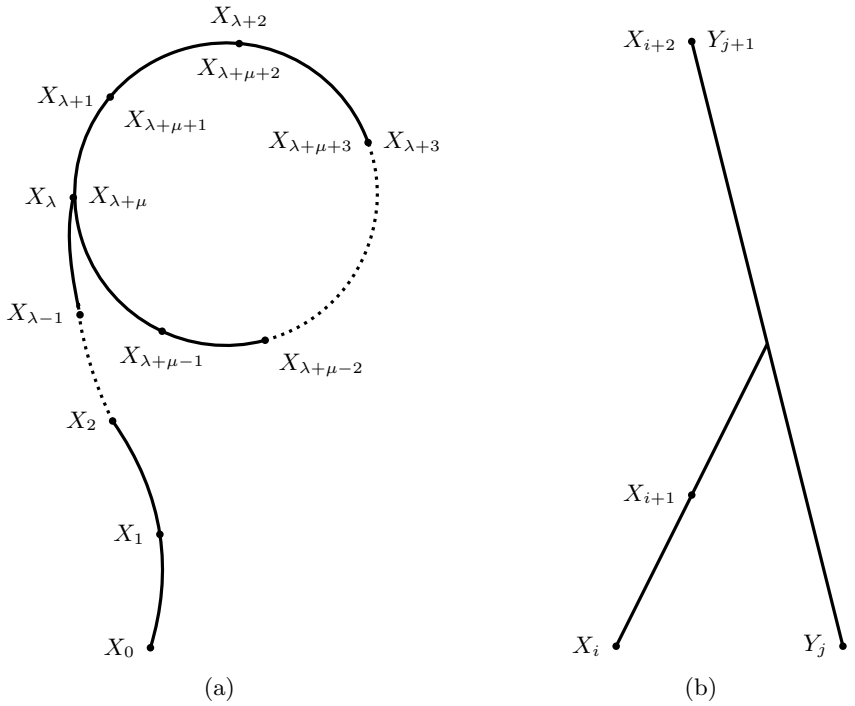

Fig. 1. *Representation of the ρ and λ shape of the single-instance Pollard rho 1(a) and the multi-instance Pollard rho method 1(b) respectively. The points X_i, Y_j represent distinguished points from two different walks.*

points $X_{i+1} = f(X_i)$, $i \geq 0$. The sequence of points represents a walk through the set of points $\langle P \rangle$. Given $X_i = c_i P + d_i Q$ and $c_i, d_i \in [0, n-1]$, f updates c_{i+1} and d_{i+1} and computes X_{i+1} as $X_{i+1} = c_{i+1} P + d_{i+1} Q$. The sequence is started from a random and known point $X_0 \in \langle P \rangle$ by selecting random values for c_0 and d_0. This sequence of points eventually collides (as operations are performed over a finite cyclic group). Let us denote λ and $\mu \geq 1$ as the smallest numbers such that $X_\lambda = X_{\lambda+\mu}$ holds. The value λ is called the tail and μ the cycle length, graphically the walk through the set of points forms a ρ shape: see Fig. 1(a). Assuming the iteration function is a random mapping of size $n = |\langle P \rangle|$, i.e. f is equally probable among all functions $F : \langle P \rangle \rightarrow \langle P \rangle$, Harris showed that the expected values of λ and μ are $\lambda = \mu = \sqrt{\frac{\pi n}{8}}$ when $n \rightarrow \infty$ [15]. The advantage of the Pollard rho method is that it uses a negligible amount of memory, by using Floyd's cycle finding method [19], compared to the baby-step-giant-step [32] method which has the same asymptotic run-time complexity.

3.3 Parallelization

In [39], van Oorschot and Wiener present a time-memory trade-off method based on the work by Quisquater and Delescaille [29]. In order to run many instances of the Pollard rho method on different processors, in order to speed up the calculation of the discrete logarithm, each instance starts with a unique value. The idea is to distinguish points in the walk using a specific property and share the distinguished points among all the processors by communicating them to a central database. Distinguished points (DTP) can be, for example, those with an x-coordinate that is divisible by 2^m, for some $m > 0$, after being normalized to $[0, p-1]$. The search for a collision among the DTPs is performed in this central database. This technique leads to a linear speed-up on the number of processors. Graphically, colliding walks form a λ shape: see Fig. 1(b).

3.4 Adding Walks

The iteration function proposed by Pollard in [28] divides $\langle P \rangle$ into three different partitions: one partition is used to double the current point while in the other two partitions a constant is added. Teske introduces in [37], based on the work by Schnorr and Lenstra [31], a class of walks for the iterating function of Pollard's rho method which achieves a similar performance, in terms of the number of iterations needed, compared to a random mapping. The main idea consists in dividing $\langle P \rangle$ into r different partitions using a partition function $h : \langle P \rangle \rightarrow [0, r-1]$.

To each partition a point is associated; for partition j the values m_j and n_j are randomly chosen in the initialization phase and $R_j = m_j P + n_j Q$ is associated with this partition. If the parallelized version of the Pollard rho method is used, the same m_j, n_j, h should be used in all instances.

The iteration function is defined as

$$X_{i+1} = f(X_i) = X_i + R_{h(X_i)}. \tag{3}$$

It is shown in [37] that values of $r \geq 16$ partitions provide performance comparable to the expected values from random mappings, overcoming a loss of approximately 20 percent of computation time that occurs when Pollard's original iteration function is used.

3.5 Montgomery's Simultaneous Inversion

Elliptic curves can be parameterized in different ways, resulting in different operation counts (cf. [1]). Since many independent walks can be processed conjointly, Montgomery's inversion technique [25], which enables to trade M inversions for $3(M-1)$ multiplications and one inversion, can be used. This places the affine Weierstrass coordinate system as the most suitable candidate. For a point addition, the cost of computing the x-coordinate is four multiplications, one squaring and $\frac{1}{M}^{\text{th}}$ inversion, when M group additions are processed in parallel. By reusing intermediate results of this computation, the y-coordinate can be computed with an additional cost of one field multiplication, see Equation (2).

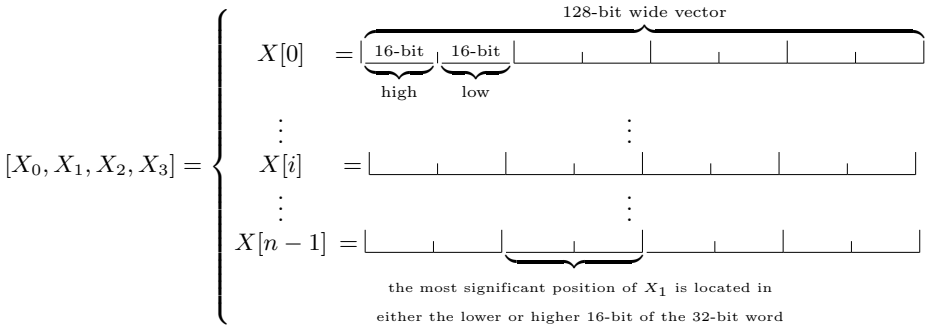

Fig. 2. *Four numbers arranged, in either all lower or higher parts.*

3.6 The Negation Map

The computation of the negative of a point $P = (x, y)$, i.e. $-P = (x, -y)$, is computationally cheap. This observation is used by Wiener and Zucherato [40] to reduce the search space by a factor of two. Given an equivalence relation $\sim$ on $\langle P \rangle$ the idea is to iterate over the set of equivalence classes $\langle P \rangle / \sim$. This can be accomplished by computing $\pm P$ and selecting the point with the smaller y-coordinate after being normalized to $[0, p-1]$. When the negation map is used, almost all equivalence classes have two elements, giving a theoretical speed-up factor of $\sqrt{2}$. This technique can be applied to all elliptic curves. In general, if most equivalence classes contain m points, the search space is reduced by a factor m. Hence, the total required number of iterations is reduced by a factor $\sqrt{m}$. Other examples of equivalence relations, aimed at anomalous binary curves, and more detailed information can be found in [40, 13, 10].

4 112-bit Elliptic Curve Domain Parameters over $\mathbb{F}_p$

As of 2009, the smallest standardized elliptic curve is over a 112-bit prime field. This elliptic curve is standardized in the Standard for Efficient Cryptography (SEC), SEC2: Recommended Elliptic Curve Domain Parameters [6] as curve *secp112r1* and in the Wireless Transport Layer Security Specification [12] as *curve number 6*.

4.1 Integer Representation in the Cell

For a high-performance implementation of arithmetic algorithms on the Cell, vectorization techniques (cf. [9]) are applied and data are represented using a 4-SIMD organization. If the radix of the number system is $r = 2^w$, with $0 < w \leq 32$, then a b-bit number is represented using $n = \lceil \frac{b}{w} \rceil$ digits. In the 4-SIMD representation, four b-bit numbers X_0, X_1, X_2, and X_3 are stored in n vectors.

Each vector $X[j]$, $0 \leq j < n$, holds four w-bit digits of the four numbers that correspond to the same digit position. The notation $[X_0, X_1, X_2, X_3]$ means that the four numbers X_0, X_1, X_2, and X_3 are grouped using 4-SIMD and operations are applied in parallel digit-wise (for the same digit positions) for all the four numbers. For modular multiplication, $w = 16$ is selected, cf. Section 4.2, and each of the n vectors is composed of four 32-bit word elements, where the 16-bit digits of four numbers are stored either in the higher or lower positions of these 32-bit word elements. Hence, each of the four b-bit numbers is represented as $X_i = \sum_{j=0}^{n-1} r^j \left(\left\lfloor \frac{X[j]}{r^{2 \cdot i + h}} \right\rfloor \bmod r \right)$ for $i \in \{0, 1, 2, 3\}$ and $h \in \{0, 1\}$, where h is 1 if data are placed in the higher bit positions and 0 otherwise. Fig. 2 depicts the data structure. For modular inversion, $w = 32$ and each of the four b-bit numbers is represented as $X_i = \sum_{j=0}^{n-1} r^j \left(\left\lfloor \frac{X[j]}{r^i} \right\rfloor \bmod r \right)$ for $i \in \{0, 1, 2, 3\}$ and adjusting the value n accordingly.

4.2 Arithmetic

The standardized elliptic curve *secp112r1* is over $\mathbb{F}_p$. Here p is prime and has the special form: $p = \frac{2^{128}-3}{11 \cdot 6949}$. In order to speed up modular multiplication and subtraction in the Pollard rho algorithm we use a redundant representation taking a larger modulus $\widetilde{p} = 2^{128} - 3 = 11 \cdot 6949 \cdot p$.

Modular Multiplication One computationally intensive operation in the point addition on the elliptic curve is modular multiplication. Furthermore, as Montgomery's simultaneous inversion technique is used to trade one modular inversion by approximately three modular multiplications, the performance of the Pollard rho algorithm highly depends on the performance of the modular multiplication.

In order to increase computation speed, operations are performed in a residue class of a larger modulus $\widetilde{p}$. This redundant representation significantly accelerates modular reduction and successive operations can be performed in this representation.

Let us define a reduction function $\mathfrak{R}$.

Definition 1. *Let $R = 2^{128}$ and $\widetilde{p} = R - 3$. Given an integer $0 \leq x < R^2$ represented in radix R; $x = x_h \cdot R + x_l$, define a map $\mathfrak{R} : \mathbb{Z}/R^2\mathbb{Z} \rightarrow \mathbb{Z}/R^2\mathbb{Z}$ such that*

$$y = \mathfrak{R}(x) = (x \bmod R) + 3 \cdot \left\lfloor \frac{x}{R} \right\rfloor$$

Note that, if $y = \mathfrak{R}(x)$ then $x \equiv y \bmod \widetilde{p}$ and $y \leq x$.

Furthermore, with high likelihood $\mathfrak{R}$ can be used to quickly reduce values modulo $\widetilde{p}$. Because $0 \leq \mathfrak{R}(x) < 4R$, for any x with $0 \leq x < R^2$, it follows that $0 \leq \mathfrak{R}(\mathfrak{R}(x)) < R + 9$. It is easily seen that $R + 9$ can be replaced by $R + 6$. Assuming that all values have more or less the same probability to occur, the result will actually most likely be $< \widetilde{p}$. Although counterexamples are simple to construct and we have no formal proof, we can confidently state the following.

Proposition 1. *For independent random* 128*-bit non-negative integers* x *and* y *there is overwhelming probability that* $0 \leq \mathfrak{R}(\mathfrak{R}(x \cdot y)) < \widetilde{p}$.

Computation of integer multiplication is performed using the data representation described in Section 4.1. In order to take advantage of the multiply-and-add instruction, we use the following property. If $0 \leq a, b, c, d < r$, then $a \cdot b + c + d < r^2$. Specifically, this property enables the addition of a 16-bit word to the result of a 16×16-bit product, used for the multi-precision multiplication and accumulation, and an extra addition of 16-bit word, which is used for carry propagation. The multi-precision products are calculated using the schoolbook method since the modulus is relatively small and the multiply-and-add instruction can be exploited. Our tests show that this approach is faster, for this particular size on this platform, compared to other methods such as Karatsuba multiplication [18].

Modular Subtraction The modular subtraction algorithm, which can be implemented as a subtraction with a conditional addition, is a basic operation. See for implementation details Section 5.1.

Modular Inversion We consider modular inversion of one positive integer x in the residue class of an odd modulus p. Taking into account the memory constrained environment of the PS3s, and the 4-SIMD organization of the SPEs, the most suitable algorithm seems to be the Montgomery algorithm for the classical modular inverse [17]. This algorithm computes modular inversion in two phases:

1. The computation of the almost Montgomery inverse $x^{-1} \cdot 2^k \bmod p$ for some known k.
2. A normalization phase where the factor $2^k \bmod p$ is removed.

In order to exploit the 4-SIMD organization, variables A_1, B_1, A_2 and B_2 are grouped and denoted as $[A_1, B_1, A_2, B_2]$. Then, the resulting 4-SIMD Extended Binary GCD algorithm is depicted in Algorithm 1.

In the algorithm, $A_1 >> t_1$ means that variable A_1 is shifted by t_1 bits towards the least significant bit position. Similarly, $B_1 << t_2$ means that variable B_1 is shifted to the most significant bit position by t_2 positions. Assignments such as $[A_1, B_1, A_2, B_2] := [A_1 >> t_1, B_1 << t_2, A_2 >> t_2, B_2 << t_1]$ mean that the four operations (shift operations in this case) are performed in parallel in SIMD. Note that, in the algorithm, operations $A_1 >> t_1$ and $A_2 >> t_2$ shift out only zero bits.

Algorithm 1 4-SIMD Extended Binary GCD

Input: $p : r^{n-1} < p < r^n$ and $\gcd(p, 2) = 1$
$x : 0 < x < r^n$ and $\gcd(x, p) = 1$

Output: $z \equiv \frac{1}{x} \bmod p$

1: $[A_1, B_1, A_2, B_2] := [p, 0, x, 1]$ and $[k_1, k_2] := [0, 0]$
2: **while** true **do**
3: /* Start of shift reduction. */
4: Find t_1 such that $2^{t_1} | A_1$
5: Find t_2 such that $2^{t_2} | A_2$
6: $[k_1, k_2] := [k_1 + t_1, k_2 + t_2]$
7: $[A_1, B_1, A_2, B_2] := [A_1 >> t_1, B_1 << t_2, A_2 >> t_2, B_2 << t_1]$
8:
9: /* Start of subtraction reduction. */
10: **if** $(A_1 > A_2)$ **then**
11: $[A_1, B_1, A_2, B_2] := [A_1 - A_2, B_1 - B_2, A_2, B_2]$
12: **else if** $(A_2 > A_1)$ **then**
13: $[A_1, B_1, A_2, B_2] := [A_1, B_1, A_2 - A_1, B_2 - B_1]$
14: **else**
15: **return** $z := B_2 \cdot (2^{-(k_1+k_2)}) \bmod p$
16: **end if**
17: **end while**

Let $g = \gcd(x, p)$ and y be a solution of $xy \equiv g \pmod{p}$. Algorithm 1 has invariants (for $j = 1, 2$)

$$\begin{aligned} A_j(2^{k_1+k_2}y) &\equiv B_j g \pmod{p}, \\ \gcd(A_1, A_2) &= g, \\ A_1B_2 - A_2B_1 &= p, \\ 2^{k_1}A_1 \le p, \quad 2^{k_2}A_2 &\le x, \\ B_1 \le 0 < B_2, \quad k_j \ge 0, \quad A_j &> 0. \end{aligned} \tag{4}$$

At line 15, a modular multiplication removes powers of 2 from the output. We can bound the exponent $k_1 + k_2$ by

$$2^{k_1+k_2} \le (2^{k_1}A_1)(2^{k_2}A_2) \le px.$$

We have $A_1 = A_2 = \gcd(A_1, A_2) = g$. If $A_2 > 1$ then we report an error to the caller. Otherwise $g = 1$. The output $z = B_2 \cdot (2^{-k_1-k_2})$ satisfies

$$z = zg \equiv B_2 \cdot (2^{-k_1-k_2})g \equiv (A_2 2^{k_1+k_2} y)2^{-k_1-k_2} \equiv A_2 y = y \pmod{p}.$$

If we pick t_1 and t_2 as large as possible during a shift reduction, then the new A_1 and A_2 will both be odd. The next subtraction and shift reductions will reduce $A_1 + A_2$ by at least a factor of 2.

The values of A_1 and A_2 are bounded by p and x, respectively. The invariant $p = A_1B_2 - A_2B_1 \ge B_2 - B_1$ bounds B_1 and B_2.

Operation	Estimated #cycles per operation	Quantity per iteration	Estimated #cycles per iteration
Modular multiplication	53	6	318
Modular subtraction	5	6	30
Montgomery reduction	24	1	24
Modular inversion	4941	$\frac{1}{400}$	12
Miscellaneous	69	1	69
Total			453

Table 1. *Estimated number of clock cycles for different operations of our Pollard rho implementation in one SPU.*

4.3 The Distinguished Point and Partition Determination

Each application of an r-adding iteration function requires the determination of the partition to follow for calculating the next point; see Section 3.4. Furthermore, the current unreduced point needs to be inspected for distinguishedness. Since we are performing arithmetic modulo $\widetilde{p}$, the coordinates of the elliptic curve point need to be reduced modulo p, i.e. this point cannot be used to uniquely determine either the partition number or the the distinguished point property. Given a point $\widetilde{P} = (\widetilde{x}, \widetilde{y})$, the idea is to compute a partial Montgomery reduction [24] instead of normalizing $\widetilde{x}$ modulo p which requires a full modular reduction at each iteration. This faster reduction computes $\widetilde{x} \cdot 2^{-16} \bmod p$, where the result of this operation is in $[0, p-1]$. The uniqueness of both the distinguished point property and the partition number is ensured.

5 Experimental Results

In this section, we present the performance analysis of our Pollard rho implementation using the techniques described in Section 4 and show how this implementation has set a record by solving a 112-bit ECDLP. The previous record in the computation of an ECDLP is for an elliptic curve over a 109-bit prime field with parameters taken from Certicom's ECC challenge [4]. That problem was solved in the year 2002 using "10^4 computers (mostly PCs) running 24 hours a day for 549 days" [5].

5.1 Implementation details

Our software implementation is optimized for the SPE-architecture of the Cell and uses all the techniques described in Section 3 with the exception of the negation map. This is because, the computational overhead for this technique, due to the conditional branches required to check for fruitless cycles [13], results (in our implementation on this architecture) in an overall performance degradation. As an iteration function a 16-adding walk is used. In order to take advantage of the Montgomery simultaneous inversion technique, 400 walks are processed

in parallel. The number of concurrent walks is adjusted to the local storage restrictions of the PS3s. At the cost of 16 counters of 32 bits each per process, updating the values c_{i+1}, d_{i+1} can be postponed until a distinguished point is found.

Note that, in our implementation, several things can go wrong: we may have dropped off the curve because we should have used curve doubling (in the unlikely case that $X_i = R_{h(X_i)}$, or in the unlikely case of incorrect reduction modulo $\widetilde{p}$, cf. Prop. 1), or a wrong point may by accident again have landed on the curve, and have nonsensical c_i, d_i values. Just as the correct iterations, these wrong points will after a while end up as distinguished points. Thus, whenever a point is distinguished, we check that it indeed lies on the curve and that the equation $X_i = c_i P + d_i Q$ holds for the alleged c_i and d_i. Only correct distinguished points are collected. If we hit upon a process that has gone off-track, all 400 concurrent processes on that SPU are terminated and restarted, each with a fresh startpoint. This type of error-acceptance leads to enormous timesavings and code-size reduction at negligible cost: we have not found even a single incorrect distinguished point during in the process of solving this ECDLP instance.

A summary of estimated clock cycles needed for each operation is detailed in Table 1. The 69 miscellaneous clock cycles stated in this table include the cost for fetching the constant for the 16-adding walk, checking if a point is distinguished and if so perform sanity checks and the overhead of conditional branches in the main and the simultaneous inversion loop.

Modular Multiplication Our implementation of the modular multiplication method, which is an 128-bit modular multiplication since we work with integers reduced modulo $\widetilde{p}$, as described in Section 4.2, is aimed at filling both the odd and even pipelines, to reduce the overall latency. The 4-SIMD multiplication is done by using the multiply-and-add instruction. Extraction of higher and lower 16-bit parts of a 32-bit word elements is done by using two shuffle operations which are performed in the odd pipeline.

Fast modular reduction, cf. Prop. 1, is implemented using eight multiply-and-add instructions, seven additions, eight extractions of the lower parts and seven extractions of the higher parts for the first reduction phase. For the second reduction, only one multiply-and-add instruction is used since the maximum number that can be added in the second reduction is 4. Most likely no further carries are generated and modular reduction is complete. This condition is checked using an conditional "if" branch with a branch-hint. In the unlikely case that carries are generated, a penalty is paid and the remaining part of the reduction code is executed.

The number of clock cycles needed for a modular multiplication is 53, as shown in Table 1. This number is an average over a long benchmark run using input data from the Pollard rho algorithm.

Modular Subtraction Modular subtraction is performed using operands represented in 4-SIMD with radix 2^{32}. A multi-word subtraction (four extended

subtractions and four generate borrow instructions), comparison (one comparison of the borrow), mask (four AND instructions) and addition (four extended additions and three extended carry generation instructions) are performed in order to avoid expensive branches. Conversions back and forth from representations using radix 2^{16} and 2^{32}, in 4-SIMD, are performed using eight shuffle operation for each conversion.

All in all, 16 instructions in the odd pipeline and 20 instructions in the even pipeline are needed for four modular subtractions. The number of clock cycles required for a single modular subtraction in practice is roughly five (see Table 1).

Modular Inversion The proposed modular inversion algorithm performs one single modular inversion using the SIMD instructions of the SPE, with either two or four active computations at a time. The 128-bit values of A_1, B_1, A_2, and B_2 are stored using the data representation described in Section 4.1 with $w = 32$. The initializations $A_1 = p$, $B_1 = 0$, $B_2 = 1$ do not depend on x. The initialization of $A_2 = x$ requires eight load and four shuffle instructions to convert the input.

A shift reduction always starts with at least one of A_1 and A_2 being odd, by Equation (4). We do not know which of these might be even, but can examine both, in a SIMD fashion. The trailing zero bit count of a positive integer A is the population count of $\overline{A} \wedge (A - 1)$. The PS3's population count instruction acts only on 8-bit data, so our t_1 and t_2 may not be maximal. The PS3 lets each vector element have its own shift or rotate count, although a single instruction cannot rotate some elements while others shift.

Within the subtraction reduction, the four 128-bit differences $A_1 - A_2$, $B_1 - B_2$, $A_2 - A_1$, and $B_2 - B_1$ are evaluated in parallel. We exit the loop if neither $A_1 - A_2$ nor $A_2 - A_1$ needs a borrow. Otherwise we update $[A_1, B_1, A_2, B_2]$ appropriately. Subtracting 1 from each element of the borrow vector gives masks of -1 or 0 depending on the sign of $A_1 - A_2$ or $A_2 - A_1$. A shuffle of these masks builds a selector which determines which parts of $[A_1, B_1, A_2, B_2]$ are updated.

The final multiplication with 2^{-k} is done by first looking up this value in a table and next computing the modular multiplication as outlined in Prop. 1. Hence, the modular inversion implementation takes as input an 128-bit integer x and outputs an 128-bit integer $z \equiv \frac{1}{x} \bmod p$ with $0 \leq x, z < \widetilde{p}$.

5.2 Performance Comparison

In the paper by Güneysu et al. [14] a field-programmable gate array (FPGA)-based multi-processing hardware architecture for the Pollard rho method targeted at elliptic curves over prime fields is described. Performance details are stated for a hardware implementation using XC3S1000 FPGAs targeted at field sizes of varying bit lengths.

Our PS3 implementation is targeted at an elliptic curve over a 112-bit prime field. We use 128-bit multiplication with fast reduction modulo the 128-bit special modulus $\widetilde{p}$. The inversion of 128-bit values is performed modulo the 112-bit

prime. Experimental results show that modular multiplication using fast reduction (see Prop. 1) is roughly 20 percent faster compared to an implementation of Montgomery multiplication on the SPE architecture. Because 128-bit reduction is used, we compare our performance results to the FPGA-results of elliptic curves over 96- and 128-bit generic prime fields [14]. These results are given for the cost-efficient parallel architecture called COPACABANA [21]. This architecture can host up to 120 FPGAs at a total cost of approximately US$ $10,000$ including material and production costs. Using this setup, a performance of $3.97 \cdot 10^7$ and $2.08 \cdot 10^7$ iterations per second can be achieved for the 96- and 128-bit versions respectively.

The current price for a PS3, as stated on large web-stores, is around US$ 300. Hence, for the price of one COPACABANA, 33 PS3s can be purchased. The resulting cluster of PS3s is able to compute $1.4 \cdot 10^9$ iterations per second. The performance results reported in [14] are dated from 2007. We scale the performance results according to Moore's law [26], i.e. the performance is doubled. The implementations by Güneysu et al. use the negation map optimization, leading to a $\sqrt{2}$ speed-up. The use of this technique results in some overhead related to the detection and handling of fruitless cycles; the reason why we decided not to use this technique in our SPE-implementation. Unfortunately, no details are given related to this overhead. After scaling the COPACABANA performance numbers by a factor two, due to Moore's law, and assuming that the negation map optimization technique is used, leading to a speed-up of a factor $\sqrt{2}$, the PS3 cluster outperforms the COPACABANA machine by a factor 12.4 and 23.8 compared to the 96- and 128-bit versions respectively.

5.3 Solving a 112-bit ECDLP

We solved the ECDLP using the parameters of the standardized curve over a 112-bit prime field using the methods and implementation as explained in this article. The expected number of iterations is $\sqrt{\frac{\pi \cdot n}{2}} \approx 8.4 \cdot 10^{16}$, where n is the prime order of the base point P as specified in the standard, assuming the negation map optimization is not used. The real number of required iterations to solve this ECDLP was only two percent higher. The calculation has been performed on a PS3 cluster of more than 200 PS3s and started on January 13, 2009 and finished on July 8, 2009. It ran on and off, occasionally interrupted by other cryptanalytic projects. When run continuously using the latest version of our code, the same calculation would have taken 3.5 months.

By selecting a DTP property with occurrence of approximately once every 2^{24} points, we needed to store $\approx 5.0 \cdot 10^9$ DTPs. Storage of a DTP $X = (x, y)$ together with the values c and d such that $X = cP + dQ$, requires $4 \cdot 112$ bits when storing in an uncompressed format. Hence, the total required storage sums up to $4 \cdot 112 \cdot 5.0 \cdot 10^9$ bits ≈ 260 gigabyte. To facilitate collision finding using standard unix commands the DTPs were stored in plaintext format increasing the required storage to 615 gigabyte.

We solved the discrete logarithm with respect to P for the point Q. The point P of order n are given in the standard and the x-coordinate of Q was chosen as $\lfloor(\pi - 3)10^{34}\rfloor$. The points P, Q and the solution to $Q = lP$ are given here:

$P =$ (188281465057972534892223778713752, 341987549103317082716786189608268 8)
$Q =$ (141592653589793238462643383279502 8, 384675960649470672428613962388554 4)
$n =$ 44516852250937147764918915425489 33
$Q =$ 312521636014772477161767351856699 $\cdot P$

6 Conclusions

We have presented a high-performance PlayStation 3 (PS3) implementation of the Pollard rho discrete logarithm algorithm on elliptic curves over prime fields. Arithmetic algorithms have been designed for the SIMD-like architectures such as the PS3. Using this implementation a record has been set by solving a 112-bit ECDLP where the parameters are taken from a standardized curve. The time required to solve this ECDLP instance is 62.6 PS3 years. This shows that given the easy accessibility and the relatively low price of these game consoles, solving ECDLPs for this bit-size is practical.

References

1. D. J. Bernstein and T. Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In *Finite Fields and Applications*, volume 461 of *Contemporary Mathematics Series*, pages 1–19, 2008.
2. J. W. Bos, N. Casati, and D. A. Osvik. Multi-stream hashing on the PlayStation 3. *PARA 2008*, 2008. To appear.
3. J. W. Bos and M. E. Kaihara. Montgomery multiplication on the Cell. *PPAM 2009*, 2009. To appear.
4. Certicom. Certicom ECC Challenge. See http://www.certicom.com/images/pdfs/cert_ecc_challenge.pdf, 1997.
5. Certicom. Press release: Certicom announces elliptic curve cryptosystem (ECC) challenge winner. See http://www.certicom.com/index.php/2002-press-releases/38-2002-press-releases/340-notre-dame-mathematician-solves-eccp-109-encryption-key-problem-issued-in-1997, 2002.
6. Certicom Research. Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters. Standard SEC2, Certicom, 2000.
7. N. Costigan and P. Schwabe. Fast elliptic-curve cryptography on the Cell broadband engine. In *Africacrypt 2009*, volume 5580 of *LNCS*, pages 368–385, 2009.
8. N. Costigan and M. Scott. Accelerating SSL using the vector processors in IBM's Cell broadband engine for Sony's Playstation 3. Cryptology ePrint Archive, Report 2007/061, 2007. http://eprint.iacr.org/.
9. B. Dixon and A. K. Lenstra. Fast massively parallel modular arithmetic. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 99–110, 1993.
10. I. M. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In *Asiacrypt 1999*, volume 1716 of *LNCS*, pages 103–121, 1999.

11. B. Flachs, S. Asano, S. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A streaming processor unit for a Cell processor. *IEEE International Solid-State Circuits Conference*, pages 134–135, February 2005.
12. W. A. P. Forum. Wireless transport layer security specification. See http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf, 2001.
13. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Improving the parallelized Pollard lambda search on anomalous binary curves. *Mathematics of Computation*, 69(232):1699–1705, 2000.
14. T. Güneysu, C. Paar, and J. Pelzl. Special-purpose hardware for solving the elliptic curve discrete logarithm problem. *ACM Transactions on Reconfigurable Technology and Systems*, 1(2):1–21, 2008.
15. B. Harris. Probability distributions related to random mappings. *The Annals of Mathematical Statistics*, 31:1045–1062, 1960.
16. H. P. Hofstee. Power efficient processor architecture and the Cell processor. In *HPCA 2005*, pages 258–262, 2005.
17. B. S. Kaliski Jr. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, 1995.
18. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. Number 145 in Proceedings of the USSR Academy of Science, pages 293–294, 1962.
19. D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, third edition, 1997.
20. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
21. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking ciphers with COPACOBANA - a cost-optimized parallel code breaker. In *CHES 2006*, volume 4249 of *LNCS*, pages 101–118, 2006.
22. A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
23. V. S. Miller. Use of elliptic curves in cryptography. In *Crypto 1985*, volume 218 of *LNCS*, pages 417–426, 1986.
24. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
25. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.
26. G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38:8, 1965.
27. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
28. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.
29. J.-J. Quisquater and J.-P. Delescaille. How easy is collision search. new results and applications to DES. In *Crypto 1989*, volume 435 of *LNCS*, pages 408–413, 1989.
30. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.
31. C. P. Schnorr and H. W. Lenstra, Jr. A Monte Carlo factoring algorithm with linear storage. *Mathematics of Computation*, 43(167):289–311, 1984.

32. D. Shanks. Class number, a theory of factorization, and genera. In *Symposia in Pure Mathematics*, volume 20, pages 415–440, 1971.
33. J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Gradute Texts in Mathematics*. Springer-Verlag, 1986.
34. M. Stevens, A. K. Lenstra, and B. de Weger. Predicting the winner of the 2008 US presidential elections using a Sony PlayStation 3. http://www.win.tue.nl/hashclash/Nostradamus/.
35. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *Crypto 2009*, volume 5677 of *LNCS*, pages 55–69, 2009.
36. O. Takahashi, R. Cook, S. Cottier, S. H. Dhong, B. Flachs, K. Hirairi, A. Kawasumi, H. Murakami, H. Noro, H. Oh, S. Onish, J. Pille, and J. Silberman. The circuit design of the synergistic processor element of a Cell processor. In *ICCAD 2005*, pages 111–117. IEEE Computer Society, 2005.
37. E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70(234):809–825, 2001.
38. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standards (DSS). FIPS-186-2, Certicom Corp., 2000. See http://csrc.nist.gov/publications/PubsFIPS.html.
39. P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.
40. M. J. Wiener and R. J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In *Selected Areas in Cryptography*, volume 1556 of *LNCS*, pages 190–200, 1998.