On the Use of the Negation Map in the Pollard Rho Method

Joppe W. Bos, Thorsten Kleinjung, and Arjen K. Lenstra

Laboratory for Cryptologic Algorithms
EPFL, Station 14, CH-1015 Lausanne, Switzerland

Abstract. The negation map can be used to speed up the Pollard rho method to compute discrete logarithms in groups of elliptic curves over finite fields. It is well known that the random walks used by Pollard rho when combined with the negation map get trapped in fruitless cycles. We show that previously published approaches to deal with this problem are plagued by recurring cycles, and we propose effective alternative countermeasures. As a result, fruitless cycles can be resolved, but the best speedup we managed to achieve is by a factor of only 1.29. Although this is less than the speedup factor of $\sqrt{2}$ generally reported in the literature, it is supported by practical evidence.

Keywords: Pollard's rho method, fruitless cycles, negation map.

1 Introduction

The difficulty of the elliptic curve discrete logarithm problem (ECDLP) underlies the security of cryptographic schemes based on elliptic curves over finite fields [11,13]. The best method known to solve ECDLP for curves without special properties is the parallelized [17] Pollard rho method [15]. A common optimization is to halve the search space by identifying a point with its inverse [18,9,7]. Because representatives for the equivalence classes can quickly be computed using the *negation map*, this equivalence relation may result in a speedup by a factor of up to $\sqrt{2}$ when solving ECDLP. For the elliptic curves over binary extension fields $\mathbf{F}_{2^t}$ from [12], order t equivalence relations can be used as well, resulting in a speedup by a factor of up to $\sqrt{2t}$ [18,9].

Usage of the negation map in the context of the Pollard rho method leads to *fruitless cycles*, useless cycles trapping the random walks. An analysis of their likelihood of occurrence appeared in [7]. Various methods have been proposed [18,9] to deal with them, all leading to costlier random walks and administrative overhead. The literature suggests that the resulting inefficiencies are negligible, and that a speedup by a factor of $\sqrt{2}$ is attainable [1, Section 19.5.5].

We analyze fruitless cycles and the previously published methods to avoid their ill effects and show that current approaches to escape from cycles suffer from *recurring cycles*. These may have contributed to the lack of practical usage of the negation map to solve prime field ECDLPs: it was not used for the solutions

G. Hanrot, F. Morain, and E. Thomé (Eds.): ANTS-IX 2010, LNCS 6197, pp. 66–82, 2010.

[10,6] of the 79-, 89-, 97- and 109-bit prime field Certicom challenges [5]. Neither was it used by the independent current 112-bit prime field record [3].

We present and analyze alternative methods to deal with fruitless cycles. All our analyses are supported by experiments. We found that the negation map indeed leads to a speedup, but we have not been able to reach more than a factor of 1.29, somewhat short of the $\sqrt{2}$ that we had hoped for. We also found that the best attainable speedup depends on the platform one uses: for instance, if the Pollard rho method is parallelized in SIMD fashion, it is a challenge to achieve any speedup at all. This has consequences for the applicability of the negation map in large scale prime field ECDLP solution attempts. For such efforts, all participating processors must use the same random walk definition, so one may desire to gear the implementation towards processors with the best performance/price ratio, such as graphics cards (which are SIMT, a SIMD variant).

The negation map (while dealing with cycles) slows down random walks in three ways. In the first place, on average more elliptic curve group operations are required per step of each walk. This is unavoidable and attempts should be made to minimize the number of additional operations. Secondly, dealing with cycles entails administrative overhead and branching, which cause a non-negligible slowdown when running multiple walks in SIMD-parallel fashion. Finally, the best way to counter the effect of the higher average number of group operations per step is making the walks "more random" by allowing a finer grained decision per step. However, the beneficial effects of this approach are, in most circumstances on current processors, wiped out by cache inefficiencies. It will be seen that it is best to strike a balance between the first and third of these slowdowns. The second slowdown somewhat affects regular PCs, but is a major obstacle to the negation map in SIMD environments.

This paper is organized as follows. Section 2 recalls background on ECDLP, the Pollard rho method and fruitless cycles. Section 3 introduces recurring cycles and presents and analyzes new methods to deal with them. Section 4 compares the various cycle reduction, detection, and escape methods in practice.

2 Preliminaries

2.1 The Elliptic Curve Discrete Logarithm Problem

Let $\mathbf{F}_p$ denote a finite field of odd prime characteristic p. Any $a, b \in \mathbf{F}_p$ with $4a^3+27b^2 \neq 0$ define an elliptic curve $E_{a,b}$ over $\mathbf{F}_p$. The additively written *group of points* $E_{a,b}(\mathbf{F}_p)$ *of* $E_{a,b}$ *over* $\mathbf{F}_p$ is defined as the *zero point* $\mathfrak{o}$ along with the set of pairs $(x, y) \in \mathbf{F}_p \times \mathbf{F}_p$ that satisfy the *shortened Weierstrass equation* $y^2 = x^3 + ax + b$. Let p, a, b and $\mathfrak{g} \in E_{a,b}(\mathbf{F}_p)$ of prime order q be such that the index $[E_{a,b}(\mathbf{F}_p) : \langle \mathfrak{g} \rangle]$ is small. For $\mathfrak{h} \in \langle \mathfrak{g} \rangle$, the ECDLP is to find an integer m such that $m\mathfrak{g} = \mathfrak{h}$. For curves without special properties, solving ECDLP is believed to require an effort on the order of $\sqrt{q}$. Pollard's rho method achieves this run time, while requiring more or less constant memory.

2.2 Pollard's Rho Method

If objects are selected truly at random and with replacement from q objects, the conditional probability at step $n+1$ of finding the first duplicate (or *collision*) is $\frac{n}{q}$ (if $n < q$). Via straightforward arguments this leads to $\sqrt{\pi q/2}$ for the expected number of steps until the first collision. If random objects are selected as $u\mathfrak{g} + v\mathfrak{h} \in \langle \mathfrak{g} \rangle$ for random integer multipliers u, v, a collision corresponds to $u, v, \bar{u}, \bar{v}$ such that $u\mathfrak{g} + v\mathfrak{h} = \bar{u}\mathfrak{g} + \bar{v}\mathfrak{h}$. Unless $\bar{v} \equiv v \bmod q$, the value $m = \frac{u-\bar{u}}{\bar{v}-v} \bmod q$ solves the discrete logarithm problem. The expected number of steps of this idealized version of Pollard's rho method [15] is $\sqrt{\pi q/2}$.

***r*-adding and *r*+*s*-mixed walks.** Pollard's rho method uses an approximation of a truly random walk in $\langle \mathfrak{g} \rangle$. Let, for a small integer r, an index function $\ell : \langle \mathfrak{g} \rangle \mapsto [0, r-1]$ induce an r-partition $\langle \mathfrak{g} \rangle = \cup_{i=0}^{r-1} \mathfrak{G}_i$ of $\langle \mathfrak{g} \rangle$, where $\mathfrak{G}_i = \{\mathfrak{x} : \mathfrak{x} \in \langle \mathfrak{g} \rangle, \ell(\mathfrak{x}) = i\}$ and all $\mathfrak{G}_i$ have cardinality close to $\frac{q}{r}$. For random integers u_i, v_i, elements $\mathfrak{f}_i = u_i\mathfrak{g} + v_i\mathfrak{h} \in \langle \mathfrak{g} \rangle$ are precomputed for $0 \le i < r$. Starting at a random but known multiple of $\mathfrak{g}$, the successor of a point $\mathfrak{p}$ of the walk is defined as $\mathfrak{p} + \mathfrak{f}_{\ell(\mathfrak{p})} \in \langle \mathfrak{g} \rangle$. It is easy to keep track of the u, v such that $\mathfrak{p} = u\mathfrak{g} + v\mathfrak{h}$.

Such an *r-adding walk* results in an expected number of steps until a collision occurs that is somewhat larger than $\sqrt{\pi q/2}$, as shown by Brent and Pollard [4] and expanded upon in [2]. Assume that ℓ is perfectly random. Let $p_i = \frac{\#\mathfrak{G}_i}{q}$. A point in the walk is said to belong to class i if its predecessor upon its first occurrence belongs to $\mathfrak{G}_i$. If the nth point belongs to $\mathfrak{G}_j$ (with probability p_j) and the $(n+1)$st point produces the first collision, the collision point cannot be of class j (this happens with probability p_j), since then the collision would have occurred in step n. Therefore, the probability that the first collision occurs at step $n+1$ is

$$\frac{n}{q}(1 - \sum_{j=0}^{r-1} p_j^2).$$

With $q' = \frac{q}{1-\sum_{j=0}^{r-1} p_j^2}$ this is $\frac{n}{q'}$. We get via the same arguments referred to above

$$\sqrt{\frac{\pi q'}{2}} = \sqrt{\frac{\pi q}{2(1 - \sum_{j=0}^{r-1} p_j^2)}} \tag{1}$$

for the expected number of steps until the first collision.

Pollard [15] uses $r = 3$, $\mathfrak{f}_0 = \mathfrak{h}$, and $\mathfrak{f}_2 = \mathfrak{g}$, but replaces the $i = 1$ case by the doubling $2\mathfrak{p}$. Teske [16] shows that a larger r, such as $r = 20$, leads to better performance on average, conform the analysis, even if none of the choices does an explicit doubling, as Pollard's $i = 1$ case.

Inclusion of doublings leads to *$r + s$-mixed walks*: with $\ell : \langle \mathfrak{g} \rangle \mapsto [0, r+s-1]$ partitioning $\langle \mathfrak{g} \rangle$ into $r+s$ parts of cardinality close to $\frac{q}{r+s}$, the next point equals $\mathfrak{p} + \mathfrak{f}_{\ell(\mathfrak{p})}$ if $0 \le \ell(\mathfrak{p}) < r$, but $2\mathfrak{p}$ if $\ell(\mathfrak{p}) \ge r$. Pollard's walk is a $2+1$-mixed walk. The analysis above applies again, assuming that we consider the doublings as one class, hit with probability p_D. Experiments by Teske show that best performance is achieved for $\frac{s}{r}$ between $\frac{1}{4}$ and $\frac{1}{2}$ but that apart from the case $r = 3$ mixed

walks are not significantly better. The analysis and our own experiments, as reported below, suggest that the optimal ratio $\frac{s}{r}$ is close to zero.

Per step the occurrence probability of the event $\mathfrak{p} = \mathfrak{f}_i$ (and thus a chance to solve the discrete logarithm problem) is negligible compared to the probability of a birthday collision. So, for r-adding walks doublings most likely will not occur.

Parallelized random walks. Parallelization of Pollard's rho method does not consist of running any number of random walks in parallel, until one of them collides: on M processors the expected speedup would be by a factor of $\sqrt{M}$, so overall it would require $\sqrt{M}$ more processing power than a single processor. The proper way to parallelize Pollard's rho method is presented in [17]. It achieves an M-fold speedup on M processors, thus requiring the same overall processing power as a single process, but in $\frac{1}{M}$th of the time. Different processes must be able to efficiently recognize if, probably at different points in time, their walks collide. To achieve this, each process generates a single random walk, each from its own random starting point, but all using the same index function ℓ and the same $\mathfrak{f}_i$'s. As soon as a walk hits upon a *distinguished point*, this point is reported. The idea is that when two walks collide – without noticing it – they will keep taking the same steps (because they use the same walk definition) and will thus both ultimately reach the same distinguished point. This will be noticed when the colliding distinguished point is reported. The discrete logarithm can then be computed from the two, hopefully distinct, pairs of integer multipliers u, v that correspond to the same distinguished point.

A distinguished point must be easy to recognize, occur with low enough probability to make it possible to store them all and to efficiently find collisions, but occur often enough for every walk to hit one. The distinguishing property could be that k specific bits of the point's x-coordinate are zero, in which case walks may hit a distinguished point once every 2^k steps.

The parallelized version of Pollard's rho method requires a unique, and thus affine, point representation to make the walks well-defined and to recognize distinguished points. The fastest suitable type of elliptic curve group arithmetic uses the affine Weierstrass point representation. Per group operation, it requires a (usually expensive) modular inversion. Its cost is amortized among the walks running in parallel per processor, at the cost of three modular multiplications per step per walk, using Montgomery's simultaneous inversion [14]. Point doubling requires an extra modular squaring compared to regular non-doubling point addition. This makes doubling on average about $\frac{7}{6}$ times slower than regular addition when parallelized walks and simultaneous inversion are used.

Using automorphisms. Following [18], define an equivalence relation $\sim$ on $\langle \mathfrak{g} \rangle$ by $\mathfrak{p} \sim -\mathfrak{p}$ for $\mathfrak{p} \in \langle \mathfrak{g} \rangle$ and, instead of searching $\langle \mathfrak{g} \rangle$ of size q, search $\langle \mathfrak{g} \rangle / \sim$ of size about $\frac{q}{2}$. Denoting the equivalence class containing $\mathfrak{p}$ and $-\mathfrak{p}$ by $\sim\mathfrak{p}$, it may be represented by the element with y-coordinate of least absolute value. It is trivial to calculate since $-(x, y) = (x, -y)$ for $(x, y) \in \langle \mathfrak{g} \rangle$. Thus, using this *negation map* one would expect to save a factor of $\sqrt{2}$ in the number of steps.

For r-adding and $r + s$-mixed walks the speedup by a factor of $\sqrt{2}$ is slightly too pessimistic. Let the definitions of p_i, p_D, and of class i be as above. Assume

Table 1. Number of steps required by the Pollard rho method in random elliptic curve groups of 31-bit prime order q over prime fields of random 31-bit prime characteristic p, divided by $\sqrt{\pi q/2}$ or by $\sqrt{\pi q/4}$ (without or with the negation map). Lowest and highest averages are over 10 measurements. Each measurement calculates the average number of steps taken until a collision occurs, over 100 000 collision searches where for each search a prime p and an elliptic curve over $\mathbf{F}_p$ are randomly selected until the order q of the group of points is prime. Overall average is the average of the 10 averages (thus, the average over one million searches). Expression (1) and (2) columns are the quotients as expected based on expressions (1) (with $p_i = \frac{1}{r}$ for $0 \le i < r$) and (2) (with $p_i = \frac{1}{r+s}$ for $0 \le i < r$ and $p_D = \frac{s}{r+s}$), respectively. Those expressions are for $q \to \infty$ and indeed for larger (smaller) q they give a better (worse) fit.

	Without negation map				With negation map			
	Averages			Expression	Averages			Expression
	lowest	overall	highest	(1)	lowest	overall	highest	(2)
8-adding	1.079	1.083	1.085	1.069	1.035	1.039	1.042	1.033
16-adding	1.032	1.037	1.040	1.033	1.015	1.017	1.020	1.016
32-adding	1.014	1.018	1.019	1.016	1.007	1.009	1.011	1.008
16 + 4-mixed	1.041	1.043	1.044	1.043	1.036	1.038	1.040	1.031
16 + 8-mixed	1.075	1.078	1.081	1.078	1.075	1.077	1.079	1.069

that the nth point belongs to $\mathfrak{G}_j$ and that the $(n+1)$st point produces the first collision while hitting the representative $\mathfrak{p}$, directly or after negation. If this step is a doubling then the analysis is as above. This happens with probability p_D^2. Otherwise, we only exclude the case that, as a result of just the addition, the two predecessors hit the same point ($\mathfrak{p}$ or $-\mathfrak{p}$). This happens with probability $\frac{p_j^2}{2}$. Therefore, the probability that the first collision occurs at step $n+1$ is

$$\frac{2n}{q}(1 - p_D^2 - \sum_{j=0}^{r-1} \frac{p_j^2}{2}).$$

As above we get

$$\sqrt{\frac{\pi q}{4(1 - p_D^2 - \frac{1}{2}\sum_{j=0}^{r-1} p_j^2)}} \tag{2}$$

for the expected number of steps until the first collision. For the same parameter values this expression is more than $\sqrt{2}$ smaller than Expression (1). However, usage of the negation map requires modifications to the iteration function due to the occurrence of *fruitless cycles*. This disadvantage of the negation map was already pointed out in [9,18]. It is the focus of this article.

The group $\langle \mathfrak{g} \rangle$ may admit other trivially computable maps. For Koblitz curves the Frobenius automorphism of a degree t binary extension field leads to a further $\sqrt{t}$-fold speedup. This does not apply to the case considered here.

Small scale experiments. We checked the accuracy of predictions based on expressions (1) and (2). The results, for 31-bit primes q, are listed in Table 1.

With all averages larger than 1, both r-adding and $r+s$-mixed walks on average perform worse than truly random walks. For most walks with the negation map the averages are lower than their negation-less counterparts, indicating that the reduction factor in the expected number of steps is indeed larger than $\sqrt{2}$. This does not imply a speedup by the same factor, because to obtain the figures costly fruitless cycle detection methods had to be used. It can be seen that $r+s$-mixed walks are disadvantageous if $s > \frac{r}{4}$.

2.3 Fruitless Cycles

Straightforward application of the negation map to Pollard's rho method with r-adding or $r+s$-mixed walks does not work due to fruitless cycles. This section describes the current state-of-the-art of dealing with those cycles.

Length 2 cycles. If a random walk step goes from $\mathfrak{p}$ to $-\mathfrak{p} - \mathfrak{f}_i$ (with probability $\frac{1}{2}$, for some i) and $-\mathfrak{p} - \mathfrak{f}_i \in \mathfrak{G}_i$ (with probability $\frac{1}{r}$), then the next point after $-\mathfrak{p} - \mathfrak{f}_i$ is $\mathfrak{p}$ again (with probability 1), thereby cancelling the effect of the previous step. It follows that a fruitless 2-cycle starts from a random point with probability $\frac{1}{2r}$, cf. [7, Proposition 31]. This 2-cycle is denoted as

$$\mathfrak{p} \xrightarrow{(i,-)} -(\mathfrak{p} + \mathfrak{f}_i) \xrightarrow{(i,-)} \mathfrak{p}.$$

Here "(i, s)" with $s \in \{-, +\}$ indicates that addition constant $\mathfrak{f}_i$ is added to a point $\mathfrak{p}$ after which the result is left as is ($s = +$) or negated ($s = -$) to find the correct representative ($\mathfrak{p} + \mathfrak{f}_i$ if $s = +$, or $-\mathfrak{p} - \mathfrak{f}_i$ if $s = -$). Any walk with two consecutive steps "$(i, -)$" is trapped in an infinite loop. Because this happens with probability $\frac{1}{2r}$, all walks can be expected to end up in fruitless cycles after a moderate number of steps when the negation map is used with r-adding walks.

Looking ahead to reduce 2-cycles. To reduce the occurrence of 2-cycles, Wiener and Zuccherato propose to use a more costly iteration function that results in a lower probability that two successive points belong to the same partition [18]. This can be achieved by using the first i of $\ell(\mathfrak{p})$, $\ell(\mathfrak{p}) + 1$, ..., $\ell(\mathfrak{p}) + r - 1$ such that $i \bmod r \neq \ell(\sim(\mathfrak{p} + \mathfrak{f}_i))$, if such an index exists (here and in the sequel indices i in $\mathfrak{f}_i$ are understood to be taken modulo r). Thus, define the next point as $f(\mathfrak{p})$ with $f : \langle\mathfrak{g}\rangle \to \langle\mathfrak{g}\rangle$ defined by

$$f(\mathfrak{p}) = \begin{cases} E(\mathfrak{p}) & \text{if } j = \ell(\sim(\mathfrak{p} + \mathfrak{f}_j)) \text{ for } 0 \leq j < r \\ \sim(\mathfrak{p} + \mathfrak{f}_i) \text{ with } i \geq \ell(\mathfrak{p}) \text{ minimal s.t. } \ell(\sim(\mathfrak{p} + \mathfrak{f}_i)) \neq i \bmod r. \end{cases}$$

The function $E : \langle\mathfrak{g}\rangle \to \langle\mathfrak{g}\rangle$ may restart the walk at a new random initial point. The latter is expected to happen once every r^r steps and will therefore not affect the efficiency. The expected cost per step of the walk is increased by a factor of $\sum_{i=0}^{r} \frac{1}{r^i}$, which lies between $1 + \frac{1}{r}$ and $1 + \frac{1}{r-1}$.

Dealing with fruitless cycles in general. Although the look-ahead technique reduces the frequency of 2-cycles, they may still occur [18]. This is elaborated upon in Section 3. Even so, it is well known that just addressing 2-cycles does

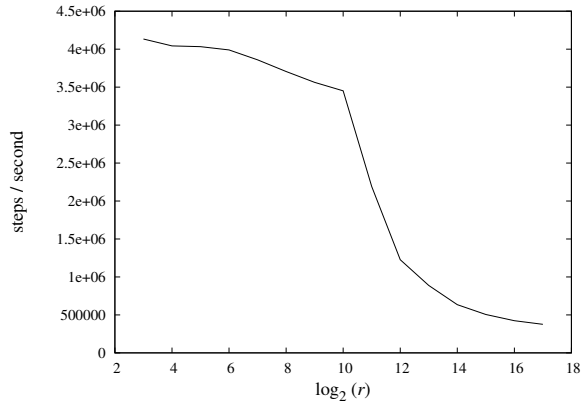

Fig. 1. Total number of steps per second as a function of r, taken by 200 parallel r-adding walks sharing the modular inversion and not using the negation map, for Pollard's rho method applied to a 131-bit prime ECDLP

not solve the problem of fruitless cycles, because longer cycles will occur as well. Reducing their occurrence requires additional overhead on top of what is already incurred to reduce 2-cycles. Given that fruitless cycles are unavoidable, they must be effectively dealt with when they occur.

In [9] a general approach is proposed to detect cycles and to escape from them: after α steps record a length β sequence of successive points and compare the next point to these β points. If a cycle is detected a cycle representative $\mathfrak{p}$ is chosen deterministically from which the cycle is escaped. One may add $\mathfrak{f}_{\ell(\mathfrak{p})+c}$ for a fixed $c \in [2, r-1]$ (the choice $c = 1$ is bad as it could lead to an immediate cycle recurrence). Instead one may add a distinct precomputed value $\mathfrak{f}'$ that does not depend on the escape-point, or one may add $\mathfrak{f}''_{\ell(\mathfrak{p})}$ from a distinct list of r precomputed values $\mathfrak{f}''_0, \mathfrak{f}''_1, \ldots, \mathfrak{f}''_{r-1}$.

In the next section we discuss fruitless cycles in greater detail and propose alternative methods that avoid problems that the method from [9] may run into.

3 Improved Fruitless Cycle Handling

The probability to enter a fruitless cycle decreases with increasing r [7]. This does not imply that it suffices to take r large enough to make the probability sufficiently low. Fig. 1 depicts the effect of increasing r-values on the performance of an r-adding walk, measured as number of steps per second. The performance deterioration can be attributed to the increasing rate of cache misses during retrieval of the addition constants $\mathfrak{f}_i$. The effect varies between processors, implementations, and elliptic curves. It is worsened for more contrived walks, such as those using the negation map where cycle reduction, detection and escape methods are unavoidable. Unless the expected overall number of steps (of order $\sqrt{q}$) is too small to be of interest, r cannot be chosen large enough to both

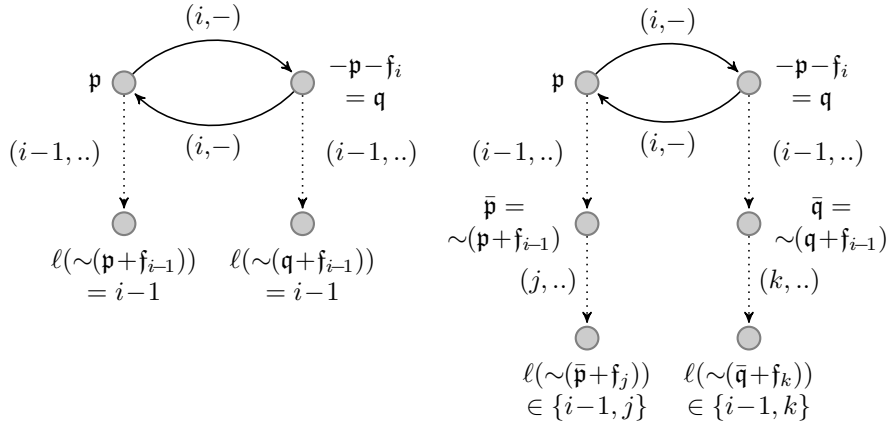

Fig. 2. 2-cycles caused by 2-cycle reduction (left) and 4-cycle reduction. The dotted steps are prevented.

avoid fruitless cycles and achieve adequate performance. Therefore, in this section we concentrate on other ways to deal with fruitless cycles. We first discuss short-cycle reduction techniques, next discuss cycle detection methods and analyze their behavior, and finally propose alternative methods.

3.1 Short Fruitless Cycle Reduction

2-cycles. Unfortunately, the look-ahead technique to reduce 2-cycles presented above introduces new 2-cycles. The dotted lines in the left example in Fig. 2 are the steps taken by the regular iteration function, the new cycle is depicted by the solid lines which are the steps taken as a result of $f(\mathfrak{p})$ and $f(\mathfrak{q})$. This new cycle occurs with probability $\frac{1}{2r^3}$. It is the most likely 2-cycle introduced by the look-ahead technique.

Lemma 1. *The probability to enter a fruitless 2-cycle when looking ahead to reduce 2-cycles while using an r-adding walk is*

$$\frac{1}{2r}\left(\sum_{i=1}^{r-1}\frac{1}{r^i}\right)^2 = \frac{(r^{r-1}-1)^2}{2r^{2r-1}(r-1)^2} = \frac{1}{2r^3} + O\left(\frac{1}{r^4}\right).$$

Proof. With i as in the definition of f, the probability is r^{-c} that $i \geq \ell(\mathfrak{p}) + c$ for $0 \leq c < r$ (considering the case $E(\mathfrak{p})$ as $i = \infty$), hence $i = \ell(\mathfrak{p}) + c$ with probability $\frac{r-1}{r}\frac{1}{r^c}$.

We compute the probability of entering a cycle consisting of points $\mathfrak{p}$ and $\mathfrak{q}$ starting at $\mathfrak{p}$. Let $j = \ell(\mathfrak{p})$ and $k = \ell(\mathfrak{q})$, and let the steps from $\mathfrak{p}$ to $\mathfrak{q}$ and back be adding $\mathfrak{f}_{j+c}$ and $\mathfrak{f}_{k+d}$, respectively. This implies that $j + c \equiv k + d \bmod r$ and that the step from $\mathfrak{p}$ to $\mathfrak{q}$ involves a negation. From the definition of f it follows

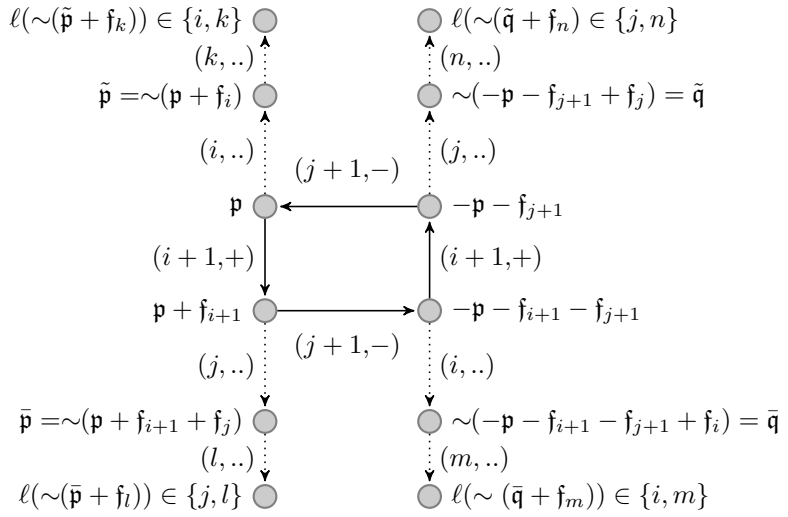

Fig. 3. A 4-cycle when the 4-cycle reduction method is used

that $\ell(\mathfrak{q}) \not\equiv j + c \bmod r$, thus $d \neq 0$ and by symmetry $c \neq 0$. Since j is given and k is determined by j, c and d, the probabilities must be summed over all possible c and d. The probability for a c, d pair is the product of the following probabilities:

- $\frac{r-1}{r}\frac{1}{r^c}$ for the first step being c;
- $\frac{1}{2}$ for the sign;
- $\frac{1}{r-1}$ for $\ell(\sim(\mathfrak{p} + \mathfrak{f}_{j+c})) = k$
  (we know already that $\ell(\sim(\mathfrak{p} + \mathfrak{f}_{j+c})) \not\equiv j + c \not\equiv k \bmod r$);
- $\frac{1}{r^d}$ for the second step being d (since $\ell(\sim(\mathfrak{q} + \mathfrak{f}_{k+d})) \not\equiv k + d \bmod r$).

This results in the probability $\dfrac{1}{2r}\displaystyle\sum_{c=1}^{r-1}\sum_{d=1}^{r-1}\frac{1}{r^c}\frac{1}{r^d}$. □

We conclude that, even when the look-ahead technique is used, 2-cycles are still too likely to occur for relevant values of q and r. Some of the new 2-cycles are prevented by other short-cycle reduction methods, but the remaining ones must be dealt with using detection and escape methods. This is discussed below.

4-cycles. Unless the addition constants $\mathfrak{f}_i$ have been chosen poorly, 3-cycles do not occur as a direct result of the negation map, so that 4-cycles are the next type of short cycles to be considered. Excluding again that the $\mathfrak{f}_i$ have unlikely properties, a fruitless 4-cycle without proper sub-cycle is of the form

$$\mathfrak{p} \xrightarrow{(i,+)} \mathfrak{p} + \mathfrak{f}_i \xrightarrow{(j,-)} -\mathfrak{p} - \mathfrak{f}_i - \mathfrak{f}_j \xrightarrow{(i,+)} -\mathfrak{p} - \mathfrak{f}_j \xrightarrow{(j,-)} \mathfrak{p}.$$

The cycle may be entered at any of its four points. Hence, a fruitless 4-cycle starts from a random point with probability $\frac{r-1}{4r^3}$. This is a lower bound for the probability of occurrence of 4-cycles when looking ahead to reduce 2-cycles.

An extension of the 2-cycle reduction method looks ahead to the first two successors of a point, thereby reducing the frequency of 2-cycles and 4-cycles, while still being deterministic:

$$g(\mathfrak{p}) = \begin{cases} E(\mathfrak{p}) & \text{if } j \in \{\ell(\mathfrak{q}), \ell(\sim(\mathfrak{q} + \mathfrak{f}_{\ell(\mathfrak{q})}))\} \text{ or } \ell(\mathfrak{q}) = \ell(\sim(\mathfrak{q} + \mathfrak{f}_{\ell(\mathfrak{q})})) \\ & \text{where } \mathfrak{q} = \sim(\mathfrak{p} + \mathfrak{f}_j), \text{ for } 0 \le j < r, \\ \mathfrak{q} = \sim(\mathfrak{p} + \mathfrak{f}_i) & \text{with } i \ge \ell(\mathfrak{p}) \text{ minimal s.t.} \\ & i \bmod r \ne \ell(\mathfrak{q}) \ne \ell(\sim(\mathfrak{q} + \mathfrak{f}_{\ell(\mathfrak{q})})) \ne i \bmod r. \end{cases}$$

Compared to $f(\mathfrak{p})$, the probability that E is called increases from $(\frac{1}{r})^r$ to at least $(\frac{2}{r})^r$ because $\ell(\sim(\mathfrak{q}+\mathfrak{f}_{\ell(\mathfrak{q})})) \in \{j \bmod r, \ell(\mathfrak{q})\}$ with probability $\frac{2}{r}$ for each j. This iteration function is at least $\frac{r+4}{r}$ times slower than the standard one, because with probability $\frac{2}{r}$ at least two additional group operations need to be carried out, an effect that is slightly alleviated by a factor of $(\frac{r-1}{r})^{\frac{1}{2}}$ since the image of g is a subset of $\langle \mathfrak{g} \rangle$ of cardinality approximately $\frac{r-1}{r}q$. The value $\sim(\mathfrak{q}+\mathfrak{f}_{\ell(\mathfrak{q})})$ can be stored for use in the next iteration. Usage of g reduces the occurrence of 4-cycles, and also prevents some of the 2-cycles newly introduced by the 2-cycle reduction method (such as the one depicted on the left in Fig. 2). But g introduces new types of 2-cycles and 4-cycles as well, both of which do indeed occur in practice. A newly introduced 2-cycle is shown in the right example in Fig. 2. There the points $\bar{\mathfrak{p}}$ and $\bar{\mathfrak{q}}$ are $\notin \mathfrak{G}_{i-1} \cup \mathfrak{G}_i$. This 2-cycle occurs with probability $\frac{2(r-2)^2}{(r-1)r^4}$, which is therefore a lower bound for the probability of 2-cycles when using the 4-cycle reduction method. Fig. 3 depicts an example of a newly introduced 4-cycle: the points reached via dotted lines belong to a partition different from their predecessors. The probability that such a 4-cycle starts from a random point is at least $\frac{4(r-2)^4(r-1)}{r^{11}}$.

We have not been able to design or to find in the literature short-cycle reduction methods that do not introduce other (lower probability) short cycles. We therefore turn our attention to cycle detection and escape methods.

3.2 Cycle Detection and Escape

Recurring cycles. The cycle detection and escape method from [9] described in Section 2.3, does not prevent recurrence to the same cycle. When using $\mathfrak{f}_{\ell(\mathfrak{p})+c}$ to escape (we fixed $c = 4$ as it worked as well as any other choice $\ne 1$), Fig. 4 depicts how the (wavy) escape from the (solid) 4-cycle recurs to the 4-cycle via one of the dotted possibilities. The probability of recurrence depends on the escape method and on which point in the cycle the walk recurs to. With $\mathfrak{f}_{\ell(\mathfrak{p})+c}$ as escape, immediate recurrence to the escape point happens with probability $\frac{1}{2r}$ when no cycle reduction is used, recurrence happens with probability at least $\frac{1}{2r^2}$ with 2-cycle reduction, and with probability at least $\frac{(r-2)^2}{r^4}$ with 4-cycle and thus 2-cycle reduction. Similar recurrences occur, with lower probabilities, when $\mathfrak{f}'$ or $\mathfrak{f}''_{\ell(\mathfrak{p})}$ are used to escape.

Lemma 2. *Lower bounds for the probabilities to enter 2-cycles or 4-cycles or to recur to cycles for three different cycle escape methods are listed in Table 2*

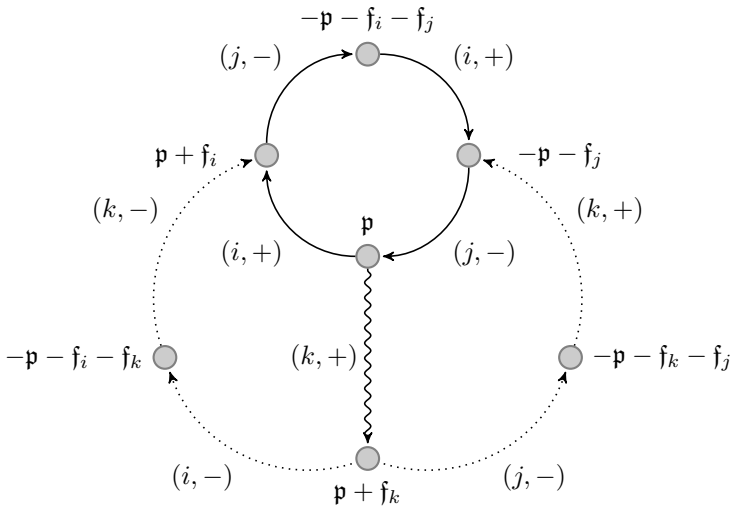

Fig. 4. Escaping from a fruitless 4-cycle, and recurring to it ($i \neq j \neq k \neq i$)

if no cycle reduction, or 2-cycle reduction (f), or 4-cycle reduction (g) is used, along with a lower bound for the slowdown factor caused by f or g.

Proof. The proofs for many entries of Table 2 were given earlier. We prove the entries in rows four and five.

Let $\mathfrak{p}$ be the escape point and let $\mathfrak{q}$ be the point it escapes to. Using $\mathfrak{f}'$ or $\mathfrak{f}''_{\ell(\mathfrak{p})}$ one can recur to the escape point $\mathfrak{p}$ by entering another cycle at $\mathfrak{q}$ and escaping from it at $\mathfrak{q}$ again. This new cycle could be a 2-cycle. For this to happen the first escape step to $\mathfrak{q}$ has to involve a negation (probability $\frac{1}{2}$), a 2-cycle has to be entered at $\mathfrak{q}$ (probabilities in first row, but see below), the escape point of this 2-cycle has to be $\mathfrak{q}$ (probability $\frac{1}{2}$), and, in the case of $\mathfrak{f}''_i$, the partition that $\mathfrak{q}$ belongs to has to be the same as the one $\mathfrak{p}$ belongs to (probability $\frac{1}{r}$). In the case of 4-cycle reduction the probability to enter a 2-cycle at $\mathfrak{q}$ is slightly lower since we do not have the information that $\ell(\sim(\mathfrak{q} + \mathfrak{f}_{\ell(\mathfrak{q})})) \neq \ell(\mathfrak{q})$; a calculation analogous to the one done at the end of Section 3.1 produces the values listed in the table. □

6-cycles. With proper $\mathfrak{f}_i$ and no sub-cycle, a common 6-cycle is of the form

$$\mathfrak{p} \xrightarrow{(i,+)} \mathfrak{p}+\mathfrak{f}_i \xrightarrow{(j,-)} -\mathfrak{p}-\mathfrak{f}_i-\mathfrak{f}_j \xrightarrow{(k,+)} -\mathfrak{p}-\mathfrak{f}_i-\mathfrak{f}_j+\mathfrak{f}_k \xrightarrow{(i,+)} -\mathfrak{p}-\mathfrak{f}_j+\mathfrak{f}_k \xrightarrow{(j,-)} \mathfrak{p}-\mathfrak{f}_k \xrightarrow{(k,+)} \mathfrak{p}$$

($i \neq j \neq k \neq i$) where with appropriate sign changes steps four and five may be swapped. It may be entered at any of its six points and occurs, when using 4-cycle reduction, with probability $\frac{1}{4r^3} + O(\frac{1}{r^4})$. A lower bound to recur to it follows by multiplying this probability with the recurring probabilities from Table 2.

Table 2. Summary of effect of cycle reduction, detection, and escape methods. With the exception of the two bold entries, all figures are lower bounds.

Cycle reduction method:		none	2-cycle	4-cycle
Probability to enter	2-cycle	$\mathbf{\frac{1}{2r}}$	$\frac{1}{2r^3}$	$\frac{2(r-2)^2}{(r-1)r^4}$
	4-cycle	$\mathbf{\frac{r-1}{4r^3}}$	$\frac{r-1}{4r^3}$	$\frac{4(r-2)^4(r-1)}{r^{11}}$
Probability to recur to escape point using	$\mathfrak{f}_{\ell(\mathfrak{p})+c}$	$\frac{1}{2r}$	$\frac{1}{2r^2}$	$\frac{(r-2)^2}{r^4}$
	$\mathfrak{f}'$	$\frac{1}{8r}$	$\frac{1}{8r^3}$	$\frac{(r-2)^2}{2r^5}$
	$\mathfrak{f}''_{\ell(\mathfrak{p})}$	$\frac{1}{8r^2}$	$\frac{1}{8r^4}$	$\frac{(r-2)^2}{2r^6}$
Slowdown factor of iteration function		n/a	$\frac{r+1}{r}$	$\frac{r+4}{r}$

3.3 Alternative Approaches

The purpose of using the negation map is to obtain a speedup, hopefully by a factor of $\sqrt{2}$. From Fig. 1 it follows that large r-values cannot be used. From Table 2 it follows that for small r-values and relevant q-values fruitless cycles are likely to occur and recur. Medium r-values look the most promising, but are not compatible with all environments.

Since fruitless cycle occurrence and recurrence cannot be rooted out, alternative methods are needed if we want to make the negation map useful. In this section several possibilities are offered.

Heuristic. *A cycle with at least one doubling is most likely not fruitless.*

Proof. Let $\mathfrak{p} = u\mathfrak{g} + v\mathfrak{h}$ be a point on the cycle. The subsequent points are obtained by adding one of the $\mathfrak{f}_i$ or by doubling, and negating if needed, thus are up to sign linear combinations of the $\mathfrak{f}_i$ and a power-of-two multiple of $\mathfrak{p}$. If $c \geq 1$ is the number of doublings in the cycle, we get a relation of the form

$$\mathfrak{p} = \pm 2^c \mathfrak{p} + \sum_{i=0}^{r-1} c_i \mathfrak{f}_i = \pm 2^c \mathfrak{p} + \sum_{i=0}^{r-1} c_i u_i \mathfrak{g} + \sum_{i=0}^{r-1} c_i v_i \mathfrak{h} \qquad \text{and thus}$$

$$\left((1 \mp 2^c)u - \sum_{i=0}^{r-1} c_i u_i\right)\mathfrak{g} + \left((1 \mp 2^c)v - \sum_{i=0}^{r-1} c_i v_i\right)\mathfrak{h} = 0,$$

where $c_i \in \mathbf{Z}$. Since $1 \mp 2^c \neq 0$, the expression $\left((1 \mp 2^c)u - \sum_{i=0}^{r-1} c_i u_i\right)$ is most likely not divisible by the group order. This also holds if $\{\mathfrak{f}_i : 0 \leq i < r\}$ is enlarged with $\mathfrak{f}'$ or with $\{\mathfrak{f}''_i : 0 \leq i < r\}$. This concludes our heuristic argument.

Cycle reduction by doubling. The regular structure required for cycles is caused by repeated addition and subtraction using the same set of constants. This structure would be broken effectively by using an occasional doubling, i.e., a mixed walk. If such walks are used, the heuristics suggest that cycles occur

only between two doublings. If the doubling frequency is sufficiently high, only short cycles would have to be dealt with.

As borne out by expressions (1) and (2) when using the idealized values $p_i = \frac{1}{r+s}$ for $0 \le i < r$ and $p_D = \frac{s}{r+s}$ for $r > 0$, and as supported by the experiments reported in Table 1, an $r+s$-mixed walk with $s > 1$ always displays noticeably less random behavior than a well-partitioned r'-adding walk for any $r' > r$. Nevertheless, using properly tuned $r+s$-mixed walks may be a way to address the cycle problem while avoiding impractically large r-values.

However, $r+s$-mixed walks have disadvantages caused by the underlying arithmetic. Given the relative speeds of addition and doubling, an $r+s$-mixed walk is $\frac{r+7s/6}{r+s}$ times slower than an r-adding walk. In a SIMD environment where many walks are processed simultaneously, per step a fraction of about $\frac{r}{r+s}$ of the walks will do an addition, whereas the others do a doubling. If the addition and doubling code differ, as is the case for the affine Weierstrass representation, the two types of steps cannot be executed simultaneously. Thus, in such environments, to avoid a slowdown by a factor of more than 2 one needs to swap walks to make all parallel step-operations identical (at non-negligible overhead), or one has to settle for a suboptimal affine point representation that allows identical code. SIMD-application of the negation map and the possibility of another point representation are subjects for further study.

Doubling based cycle reduction and escape. Taking into account that doubling should not be used too frequently, usage could be limited to cycle reduction or escape. This would not solve the SIMD-issue, but the relative inefficiency and non-randomness would be addressed. If doublings are used to escape from fruitless cycles, they would not recur, as that would contradict the heuristics. Cycle reduction using doubling replaces $f(\mathfrak{p})$ and $g(\mathfrak{p})$ by $\bar{f}(\mathfrak{p})$ and $\bar{g}(\mathfrak{p})$, respectively, where

$$\bar{f}(\mathfrak{p}) = \begin{cases} \sim(\mathfrak{p} + \mathfrak{f}_{\ell(\mathfrak{p})}) & \text{if } \ell(\mathfrak{p}) \neq \ell(\sim(\mathfrak{p} + \mathfrak{f}_{\ell(\mathfrak{p})})), \\ \sim(2\mathfrak{p}) & \text{otherwise,} \end{cases}$$

$$\bar{g}(\mathfrak{p}) = \begin{cases} \mathfrak{q} = \sim(\mathfrak{p} + \mathfrak{f}_{\ell(\mathfrak{p})}) & \text{if } \ell(\mathfrak{q}) \neq \ell(\mathfrak{p}) \neq \ell(\sim(\mathfrak{q} + \mathfrak{f}_{\ell(\mathfrak{q})})) \neq \ell(\mathfrak{q}), \\ \sim(2\mathfrak{p}) & \text{otherwise.} \end{cases}$$

It follows from the heuristics that these functions avoid recurring fruitless cycles.

Alternative cycle detection. Because shorter cycles are more frequent, a potentially interesting modification of the cycle detection method from [9] (described at the end of Section 2.3) would be to occasionally compare a point to its kth successor, where k is the least common multiple of all even short cycle lengths that one wants to catch. Detecting, for instance, cycles up to length 12 requires only $\frac{1}{120}$th comparison per step. This can be done in several steps, recording every 12th point to catch 4- and 6-cycles, recording every 10th of these recorded points to catch 8- and 10-cycles, etc. It can be combined with the regular method with large α and β to catch longer cycles infrequently.

However, if a cycle has been detected the k points need to be recorded as before, so an escape point can be chosen deterministically. This argues against

using large k. It also suggests that an improvement can be expected only if cycles occur with low probability, and therefore that the improvement will be marginal at best (cf. α and β choices in Section 4). For this reason we did not conduct extensive experiments with this method.

4 Comparison

We implemented and compared on a traditional non-SIMD platform all previously published and newly proposed methods to deal with fruitless cycles when using the negation map. Here we report on our findings. It quickly turned out that the cycle detection methods from [9] when combined with doubling based cycle reduction and escape, are considerably more efficient than $r+s$-mixed walks with their on average slower steps and less random behavior. Mixed walks are therefore not further discussed. Experiments with the alternative cycle detection method were quickly abandoned as well.

For each combination of iteration function, escape method, and r-value a search was conducted to determine the α and β to be used for the cycle detection method from [9]. Using a heuristic argument that for $\beta = 2k$ with k much smaller than r, cycles of length $\geq \beta$ occur with probability on the order of $\frac{(k-1)!}{(2r)^k}$, values for k that make this probability low enough resulted in good initial values for the search for close to optimal α and β. To give some examples, for "f, e," as explained in Table 3 we used $\alpha = 31$ and $\beta = 20$ for $r = 16$, $\alpha = 3264$ and $\beta = 12$ for $r = 128$, and $\alpha = 52\,418$ and $\beta = 10$ for $r = 256$. For "$\bar{f}$, $\bar{\text{e}}$" and the same r-values we used the same β-values but replaced the α-values by 1 618, 838 848, and 53 687 081, respectively.

Each of the benchmarks presented in Table 3 was run on a single core of an AMD Phenom 2.2GHz 4-core processor, with each of the four cores processing a different combination. A 10-bit distinguishing property was used to get a significant amount of data in a reasonable amount of time. This somewhat affects the performance, but not the cycle behavior as walks continue after hitting a distinguished point. The figures in millions as given in the table are thus an underestimate for the actual per-core yield in units when a more realistic 30-bit distinguishing property would be used (since $2^{30}/2^{10} = 2^{20} \approx 10^6$).

In order to be able to compare the long term yield figures, the expected number of steps must be taken into account using expressions 1 and 2. As a result, the yields are corrected by a factor of $(\frac{r-1}{r})^{\frac{1}{2}}$ for the iteration functions that do not use the negation map, and by a factor of $(\frac{2r-1}{r})^{\frac{1}{2}}$ for the others, with an extra factor of $(\frac{r}{r-1})^{\frac{1}{2}}$ for g and $\bar{g}$. After this correction, the best iteration function without the negation map is the one with $r = 64$. Comparing that one with each iteration function that uses the negation map, thus boosting the latter's yield ratio by a factor of $C = ((\frac{2r-1}{r})/(\frac{63}{64}))^{\frac{1}{2}}$ or $C = ((\frac{2r-1}{r-1})/(\frac{63}{64}))^{\frac{1}{2}}$ for g and $\bar{g}$, leads to the long term speedup figure given in Table 3. Note that the correction factor C depends on the iteration function, and is close to and for some r larger than $\sqrt{2}$.

Table 3. For the (iteration function, escape method, r-value) combinations specified, the non-italics entries list the long term yield (millions of distinguished points, found during the second half hour) and the long term speedup over the best r-value ($r = 64$) without the negation map, taking into account the correction factor C as explained in the text. Cycle detection and subsequent escape by adding $\mathfrak{f}_{\ell(\mathfrak{p})+4}$, $\mathfrak{f}'$, $\mathfrak{f}''_{\ell(\mathfrak{p})}$ and by doubling is indicated by "e," "e′," "e″" and by "ē," respectively. The iteration functions f (2-cycle reduction), g (4-cycle and 2-cycle reduction), $\bar{f}$ (2-cycle reduction using doubling), and $\bar{g}$ (4-cycle and 2-cycle reduction using doubling) are as in sections 2.3, 3.1 and 3.3. The yields are for 256 parallel walks (sharing the inversion) for a 131-bit ECDLP with a 131-bit prime order group. The yields during the first half hour are almost consistently higher, considerably so for poorly performing combinations. They are not meaningful and are thus not listed. The italics entries are A above D, followed by the maximal achievable speedup factor of $\frac{C(10^9-A)}{10^9+D/6}$, as explained in the text.
†: This applies to "no reduction, no escape," "just f," "just $\bar{f}$," "just e," and "just e′."

	$r = 16$	$r = 32$	$r = 64$	$r = 128$	$r = 256$	$r = 512$
Without negation map						
	7.29: 0.98	7.28: 0.99	**7.27**: 1.00	7.19: 0.99	6.97: 0.96	6.78: 0.94
With negation map						
†	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.00: 0.00
just g	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.04: 0.01	3.59: 0.70
just $\bar{g}$	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.75: 0.15	4.90: 0.96	5.90: 1.16
just e″	0.00: 0.00	0.00: 0.00	0.00: 0.00	0.61: 0.12	4.94: 0.97	5.73: 1.12
just ē	3.34: 0.64	4.89: 0.95	5.85: 1.14	6.10: 1.19	6.28: 1.23	6.18: 1.21
f, e	0.00: 0.00	0.00: 0.00	1.52: 0.30	5.93: 1.16	6.47: 1.27	6.36: 1.25
	$^{9.4e8}_{0.0e0}\}0.08$	$^{6.6e8}_{0.0e0}\}0.48$	$^{1.0e8}_{0.0e0}\}1.28$	$^{3.6e7}_{0.0e0}\}1.37$	$^{2.9e7}_{0.0e0}\}1.38$	$^{2.5e7}_{0.0e0}\}1.39$
f, e′	0.00: 0.00	3.24: 0.63	6.04: 1.18	6.41: 1.25	6.29: 1.23	6.21: 1.22
	$^{3.9e8}_{0.0e0}\}0.86$	$^{8.0e7}_{0.0e0}\}1.30$	$^{4.6e7}_{0.0e0}\}1.35$	$^{3.3e7}_{0.0e0}\}1.38$	$^{2.9e7}_{0.0e0}\}1.38$	$^{2.6e7}_{0.0e0}\}1.39$
f, e″	0.00: 0.00	5.34: 1.04	6.21: 1.21	6.30: 1.23	6.20: 1.21	5.99: 1.17
	$^{1.3e8}_{0.0e0}\}1.22$	$^{6.0e7}_{0.0e0}\}1.33$	$^{4.2e7}_{0.0e0}\}1.36$	$^{3.3e7}_{0.0e0}\}1.38$	$^{2.9e7}_{0.0e0}\}1.38$	$^{2.7e7}_{0.0e0}\}1.39$
f, ē	3.71: 0.72	6.36: 1.24	6.50: 1.27	6.57: **1.29**	6.47: 1.27	6.30: 1.25
	$^{9.2e7}_{9.9e5}\}1.27$	$^{6.8e7}_{2.8e5}\}1.32$	$^{4.2e7}_{6.5e4}\}1.36$	$^{3.3e7}_{1.5e4}\}1.38$	$^{2.9e7}_{3.8e3}\}1.38$	$^{2.7e7}_{9.7e2}\}1.39$
g, e	0.00: 0.00	0.01: 0.00	4.89: 0.96	6.22: 1.22	6.23: 1.22	6.05: 1.19
	$^{8.7e8}_{0.0e0}\}0.19$	$^{3.7e8}_{0.0e0}\}0.91$	$^{6.6e7}_{0.0e0}\}1.34$	$^{4.2e7}_{0.0e0}\}1.37$	$^{3.3e7}_{0.0e0}\}1.38$	$^{1.3e7}_{0.0e0}\}1.41$
g, e′	0.00: 0.00	0.01: 0.00	5.32: 1.05	6.26: 1.23	6.25: 1.23	6.11: 1.20
	$^{7.8e8}_{0.0e0}\}0.32$	$^{3.0e8}_{0.0e0}\}1.00$	$^{6.0e7}_{0.0e0}\}1.35$	$^{4.1e7}_{0.0e0}\}1.37$	$^{3.0e7}_{0.0e0}\}1.38$	$^{5.5e7}_{0.0e0}\}1.35$
g, e″	0.00: 0.00	1.09: 0.21	5.37: 1.13	6.08: 1.20	6.06: 1.19	5.86: 1.15
	$^{7.6e8}_{0.0e0}\}0.34$	$^{1.2e8}_{0.0e0}\}1.27$	$^{6.0e7}_{0.0e0}\}1.35$	$^{4.2e7}_{0.0e0}\}1.37$	$^{3.5e7}_{0.0e0}\}1.38$	$^{4.3e7}_{0.0e0}\}1.37$
g, ē	0.76: 0.15	5.91: 1.17	6.02: 1.18	6.25: 1.23	6.13: 1.20	6.00: 1.18
	$^{3.3e8}_{1.6e5}\}0.97$	$^{1.7e8}_{6.0e4}\}1.19$	$^{8.1e7}_{8.1e3}\}1.32$	$^{5.4e7}_{1.0e3}\}1.35$	$^{4.0e7}_{1.2e2}\}1.37$	$^{2.7e7}_{9.0e0}\}1.39$
$\bar{f}$, e	0.00: 0.00	0.00: 0.00	2.70: 0.53	5.96: 1.16	6.34: 1.24	6.20: 1.21
	$^{8.7e8}_{2.4e6}\}0.18$	$^{4.3e8}_{1.7e7}\}0.80$	$^{5.4e7}_{1.5e7}\}1.34$	$^{1.1e7}_{7.7e6}\}1.41$	$^{1.0e7}_{3.9e6}\}1.41$	$^{1.4e7}_{1.9e6}\}1.40$
$\bar{f}$, e′	0.01: 0.0	4.24: 0.82	6.32: 1.23	6.43: 1.26	6.33: 1.24	6.20: 1.22
	$^{2.6e8}_{4.3e7}\}1.03$	$^{6.8e7}_{2.9e7}\}1.31$	$^{3.9e7}_{1.5e7}\}1.36$	$^{3.2e7}_{7.6e6}\}1.38$	$^{2.8e7}_{3.8e6}\}1.38$	$^{2.7e7}_{1.9e6}\}1.39$
$\bar{f}$, e″	1.34: 0.26	5.80: 1.13	6.23: 1.22	6.21: 1.22	6.15: 1.20	6.00: 1.18
	$^{8.9e7}_{5.2e7}\}1.27$	$^{5.3e7}_{2.9e7}\}1.33$	$^{3.9e7}_{1.5e7}\}1.36$	$^{3.6e7}_{7.5e6}\}1.37$	$^{2.8e7}_{3.8e6}\}1.38$	$^{2.6e7}_{1.9e6}\}1.39$
$\bar{f}$, ē	5.58: 1.06	6.14: 1.18	6.34: 1.23	6.42: 1.25	6.27: 1.23	6.07: 1.19
	$^{6.1e7}_{4.2e7}\}1.31$	$^{3.7e7}_{3.0e7}\}1.36$	$^{1.8e7}_{1.5e7}\}1.39$	$^{1.1e7}_{7.7e6}\}1.41$	$^{1.0e7}_{3.9e6}\}1.41$	$^{1.4e7}_{1.9e6}\}1.40$
$\bar{g}$, e	2.56: 0.51	5.80: 1.15	6.02: 1.18	6.09: 1.20	6.19: 1.21	5.74: 1.13
	$^{1.4e8}_{9.9e7}\}1.23$	$^{7.9e7}_{5.6e7}\}1.31$	$^{5.1e7}_{2.9e7}\}1.35$	$^{4.1e7}_{1.5e7}\}1.37$	$^{2.6e7}_{7.6e6}\}1.39$	$^{7.7e6}_{3.9e6}\}1.41$
$\bar{g}$, e′	4.74: 0.94	5.88: 1.16	6.14: 1.21	6.28: 1.23	6.05: 1.19	5.80: 1.14
	$^{1.2e8}_{1.0e8}\}1.25$	$^{7.8e7}_{5.6e7}\}1.31$	$^{5.3e7}_{2.9e7}\}1.35$	$^{3.9e7}_{1.5e7}\}1.37$	$^{2.6e7}_{7.6e6}\}1.39$	$^{7.7e6}_{3.9e6}\}1.41$
$\bar{g}$, e″	4.72: 0.94	5.80: 1.15	6.08: 1.20	6.05: 1.19	5.91: 1.16	5.67: 1.11
	$^{1.2e8}_{1.0e8}\}1.25$	$^{7.7e7}_{5.6e7}\}1.31$	$^{5.3e7}_{2.9e7}\}1.35$	$^{3.8e7}_{1.5e7}\}1.37$	$^{1.8e7}_{7.6e6}\}1.40$	$^{7.7e6}_{3.9e6}\}1.41$
$\bar{g}$, ē	4.83: 0.96	5.87: 1.16	6.09: 1.20	6.16: 1.21	6.09: 1.20	5.70: 1.12
	$^{1.2e8}_{1.0e8}\}1.25$	$^{7.9e7}_{5.6e7}\}1.31$	$^{5.2e7}_{2.9e7}\}1.35$	$^{4.0e7}_{1.5e7}\}1.37$	$^{2.6e7}_{7.6e6}\}1.39$	$^{7.7e6}_{3.9e6}\}1.41$

Non-doubling 2-cycle reduction (f) with doubling-based cycle escape ($\bar{e}$) and $r = 128$ performed best, with an overall speedup by a factor of 1.29: although fewer distinguished points are found than for the best case without the negation map ($r = 64$), there is a considerable overall gain because fewer distinguished points (by a factor of C, for the relevant C) should suffice. For $r = 16$ most iteration functions with the negation map perform poorly.

We measured to what extent our failure to achieve a speedup by a factor of $\sqrt{2}$ can be blamed on cycle detection and escape and other overheads, and which part is due to the higher average cost of the iteration function. For most combinations in Table 3 we counted the number S of *useful* steps performed when doing 10^9 group operations, while keeping track of the number D of doublings among them. Here a step is useful if it is not taken as part of a fruitless cycle, so all D doublings are useful. Without the negation map, S would be 10^9 and $D = 0$; this is the basis for the comparison. With the negation map, $A = 10^9 - S$ is counted as the number of *additional* additions due to cycle reductions or fruitless cycles. The inherent slowdown of that iteration function is then $1 + \frac{A+D/6}{S}$, so that it can achieve a speedup by a factor of at most $\frac{CS}{S+A+D/6} = \frac{C(10^9-A)}{10^9+D/6}$, with C as defined above.

Based on Table 3 and Fig. 1, we conclude that our failure to better approach the optimal speedup by a factor of $\sqrt{2}$ is due to an onset of cache effects combined with various overheads. The italics figures from Table 3 make us believe that improvements may be obtained when using better implementations.

Previous results. The only publication that we know that presents practical data about Pollard's rho method used with the negation map is [8]. Only relatively small ECDLPs were solved (42- and 43-bit prime fields) and small r-values were avoided. The adverse cycle behavior that we witnessed can therefore not be expected and we doubt if the results reported are significant for the sizes that we consider. Only mixed walks were used, and an overall speedup by a factor of about 1.35 was reported. Cycle escaping was done by jumping to the sum of all points in a cycle, which cannot be expected to work in general because the sum may depend just on the addition constants.

5 Conclusion

With judicious application of doubling, usage of the negation map to solve ECDLPs over prime fields using Pollard's rho method can indeed be recommended. In the best of circumstances that we have been able to create, however, the speedup falls short of the hoped for $\sqrt{2}$, but is with 1.29 still considerable.

This conclusion does not apply to SIMD-environments where occasional doublings cause considerable delays. Alternative point representations need to be considered to assess the usefulness of the negation map for SIMD platforms, in particular because such platforms are becoming popular again.

Acknowledgements. This work was supported by the Swiss National Science Foundation under grant numbers 200021-119776 and 206021-117409 and

by EPFL DIT. We gratefully acknowledge useful suggestions by Marcelo E. Kaihara and very insightful comments by the ANTS reviewers.

References

1. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC (2006)
2. Bailey, D.V., et al.: Breaking ECC2K-130. In: Cryptology ePrint Archive, Report 2009/541 (2009), http://eprint.iacr.org/
3. Bos, J.W., Kaihara, M.E., Montgomery, P.L.: Pollard rho on the PlayStation 3. In: Workshop record of SHARCS 2009, pp. 35–50 (2009), http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf
4. Brent, R.P., Pollard, J.M.: Factorization of the eighth Fermat number. Math. Comp. 36(154), 627–630 (1981)
5. Certicom. Certicom ECC Challenge (1997), http://www.certicom.com/images/pdfs/cert_ecc_challenge.pdf
6. Certicom. Press release: Certicom announces elliptic curve cryptosystem (ECC) challenge winner (2002), 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
7. Duursma, I.M., Gaudry, P., Morain, F.: Speeding up the discrete log computation on curves with automorphisms. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 103–121. Springer, Heidelberg (1999)
8. Escott, A.E., Sager, J.C., Selkirk, A.P.L., Tsapakidis, D.: Attacking elliptic curve cryptosystems using the parallel Pollard rho method. CryptoBytes Technical Newsletter 4(2), 15–19 (1999), ftp.rsasecurity.com/pub/cryptobytes/crypto4n2.pdf
9. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Improving the parallelized Pollard lambda search on anomalous binary curves. Math. Comp. 69(232), 1699–1705 (2000)
10. Harley, R.: Elliptic curve discrete logarithms project, http://pauillac.inria.fr/~harley/
11. Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48, 203–209 (1987)
12. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
13. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
14. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. 48, 243–264 (1987)
15. Pollard, J.M.: Monte Carlo methods for index computation (mod p). Math. Comp. 32, 918–924 (1978)
16. Teske, E.: On random walks for Pollard's rho method. Math. Comp. 70(234), 809–825 (2001)
17. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology 12(1), 1–28 (1999)
18. Wiener, M.J., Zuccherato, R.J.: Faster attacks on elliptic curve cryptosystems. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 190–200. Springer, Heidelberg (1999)