Speeding Up the Pollard Rho Method on Prime Fields

Jung Hee Cheon, Jin Hong, and Minkyu Kim

ISaC and Department of Mathematical Sciences
Seoul National University, Seoul 151-747, Korea
{jhcheon,jinhong,minkyu97}@snu.ac.kr

Abstract. We propose a method to speed up the r-adding walk on multiplicative subgroups of the prime field. The r-adding walk is an iterating function used with the Pollard rho algorithm and is known to require less iterations than Pollard's original iterating function in reaching a collision. Our main idea is to follow through the r-adding walk with only partial information about the nodes reached.
The trail traveled by the proposed method is a normal r-adding walk, but with significantly reduced execution time for each iteration. While a single iteration of most r-adding walks on $\mathbf{F}_p$ require a multiplication of two integers of $\log p$ size, the proposed method requires an operation of complexity only linear in $\log p$, using a pre-computed table of size $O((\log p)^{r+1} \cdot \log\log p)$. In practice, our rudimentary implementation of the proposed method increased the speed of Pollard rho with r-adding walks by a factor of more than 10 for 1024-bit random primes p.
keywords: Pollard rho, r-adding walk, discrete logarithm problem, prime field

1 Introduction

Let G be a finite cyclic group of order q generated by g. Given $h \in G$, the discrete logarithm problem (DLP) over G is to find the smallest non-negative integer x such that $g^x = h$. The answer x is called the discrete logarithm of h to the base g, and is denoted by $\log_g h$. Along with the integer factorization problem, the DLP is one of two most important mathematical primitives in public key cryptography and its hardness is the basis of various cryptosystems such as Diffie-Hellman key agreement protocol [3], ElGamal cryptosystem [6], and signature schemes [5, 6].

Many of these systems, including the Digital Signature Standard [5], are implemented on a multiplicative subgroup G of prime order q of a prime field $\mathbf{F}_p$. In such a setting, the index calculus method [1] determines the size of p to be used, but the size of q is set by the Pollard rho method [14].

In this work, we use the r-adding walk style of iterating function for the Pollard rho method, which is known to require less iterations before collision than Pollard's original iterating function. In an r-adding walk, a set $\mathcal{M}$ of r random elements from G is first fixed. Given the i-th element $g_i \in G$ of the

walk, the $(i+1)$-th element g_{i+1} is defined to be the product of g_i and an element $M_s \in \mathcal{M}$, whose choice is given by the *index* $s = s(g_i)$, a function of g_i. Our idea is to define the index function s in such a way that $s(g_{i+1})$ can be computed from g_i and $M_{s(g_i)}$, without fully computing the product $g_{i+1} = g_i M_{s(g_i)}$. In the next iteration, $g_{i+2} = g_{i+1} \cdot M_{s(g_{i+1})}$ is considered as a product of g_i and $M_{s(g_i)} M_{s(g_{i+1})}$, with the second term taken from a pre-computed table of products among $\mathcal{M}$ elements. Thus, $s(g_{i+2})$ is computed without fully computing g_{i+2}. More generally, we prepare a table $\mathcal{M}_\ell := (\mathcal{M} \cup \{1\})^\ell$ of ℓ products from $\mathcal{M}$ and a full product computation is done when we reach ℓ iterations. Our method can be used with the distinguished points [15] collision detection method, and hence allows efficient parallelization, as with the original Pollard rho, i.e., n times speedup with n processors [12].

The proposed method produces a normal r-adding walk trail, and hence should reach a collision and solve DLP in the same number of steps as with any other r-adding walk, but the execution time of each iteration is significantly reduced. For 1024-bit random primes p the proposed algorithm replaces a multiplication of two 1024-bit words by 64 multiplications between a 16-bit word and a 32-bit word, and our rudimentary implementation of the proposed method was faster than the usual r-adding walks by a factor of more than 10. An incremental use of this algorithm will reduce each iteration of the original r-adding walk on $G \subset \mathbf{F}_p^\times$ from one multiplication of integers of $\log p$ size to an operation of complexity linear in $\log p$, using a pre-computed table of size $O((\log p)^{r+1} \cdot \log\log p)$.

Previous Works The fastest algorithm for the DLP on a finite field $\mathbf{F}_p$ is the index calculus method whose complexity is sub-exponential in the size of the base field [1]. Since the performance of the method depends on the size of the base field, this method has the same performance on any subgroup of $\mathbf{F}_p^\times$. If the subgroup has a composite order, we can use Pohlig and Hellman [13] algorithm to reduce the DLP in the subgroup to the DLP in its prime order subgroups.

For prime-order cyclic groups G, including multiplicative subgroups of sufficiently large finite fields, the first non-trivial algorithm solving the DLP was the Baby-Step Giant-Step method suggested by Shanks [20]. It requires $O(\sqrt{q})$ operations and memory to work on an abelian group of order q. Pollard [14] proposed a probabilistic algorithm, called the Pollard rho method, with the same complexity, but requiring only small size of memory. There have been several variants proposing different collision detection methods [2, 11, 19] and iterating functions [17, 23]. An efficient parallelization of Pollard rho was developed by van Oorschot and Wiener [12] using distinguished points. For (hyper-)elliptic curves with fast endomorphisms, more efficient variants of Pollard rho methods are known [4, 7, 25].

Organization In Section 2, we introduce the Pollard rho method, r-adding walks, and the distinguished point collision detection method. In Section 3, we propose *Tag Tracing*, a method to speed up Pollard rho. In Section 4, we apply

this to prime fields and analyze its complexity. Also, we present some implementation result for 1024-bit primes. In Section 5, we estimate the asymptotic complexity of our algorithm when it is used incrementally. Section 6 concludes this paper. Tag tracing on binary fields is briefly treated in Appendix B.

2 Pollard Rho Algorithm

To set the basis of our discussion and fix notation, we will quickly review variants of the Pollard rho method in this section. Readers should consult the original papers for any detail. Throughout this paper $G = \langle g \rangle$ will be a finite cyclic group of prime order q, on which we wish to solve a discrete logarithm problem.

2.1 Function iteration and collision

Given any function $f : G \to G$, we can create a sequence $(g_i)_{i \geq 0}$ by iteratively defining

$$g_{i+1} = f(g_i) \quad (i \geq 0),$$

starting from a random starting point $g_0 \in G$. Because G is a finite set, this sequence is eventually periodic. The smallest integers $\mu \geq 0$ and $\lambda \geq 1$ satisfying $g_{\lambda+\mu} = g_\mu$ are said to be the pre-period and period of the sequence $(g_i)_{i \geq 0}$, respectively.

When the function f is chosen uniformly at random from the set of all functions sending G to G, the value $\lambda + \mu$ is expected to be $\sqrt{\pi q/2} \sim 1.253\sqrt{q}$. Each variant of Pollard rho method provides an *iterating function* f and a method to detect a *collision*, i.e., the happening of $g_i = g_j$ with $i \neq j$.

Suppose we are trying to solve for $\log_g h$. Given any element $y \in G$, there are many ways to write it in the exponent form $y = g^a h^b$. Let us say that a function $f : G \to G$ is *exponent traceable*, or *allows exponent tracing*, with respect to g and h, if it is possible to express the function in the form

$$f(g^a h^b) = g^{f_g(a,b)} h^{f_h(a,b)},$$

with some (simple) functions f_g and f_h of the exponents. For example, if f was the squaring function on G, we could set $f_g(a, b) = 2a$ and $f_h(a, b) = 2b$.

The iterating function of a Pollard rho algorithm variant is always chosen in such a way that it is exponent traceable. Thus, starting from $g_0 = g^{a_0} h^{b_0}$, with randomly chosen, but known, (a_0, b_0), we can always keep track of the exponents (a_i, b_i) satisfying $g_i = g^{a_i} h^{b_i}$. Then, when a collision $g_i = g_j$ is detected, setting $x = \log_g h$, we know $g^{a_i}(g^x)^{b_i} = g^{a_j}(g^x)^{b_j}$, so we can use

$$a_i + x \cdot b_i \equiv a_j + x \cdot b_j \pmod{q}$$

to solve for x.

2.2 Iterating functions

An iterating function is taken to be of good design if the number of iterations it takes to reach a collision is close to $\sqrt{\pi q/2}$, the value expected of a random function.

Pollard Pollard [14] originally targeted the DLP on $(\mathbf{Z}/p\mathbf{Z})^*$, but his iterating function, which we shall denote as $f_P : G \to G$, can be modified for use on any cyclic group. Let $G = T_0 \cup T_1 \cup T_2$ be a partition of G into nearly equal sized subsets. The iterating function is defined as follows.

$$f_P(y) = \begin{cases} gy, & \text{if } y \in T_0, \\ y^2, & \text{if } y \in T_1, \\ hy, & \text{if } y \in T_2. \end{cases}$$

It is clear that this allows exponent tracing. For example, when $g^a h^b \in T_0$, we have $(f_P)_g(a, b) = a+1$ and $(f_P)_h(a, b) = b$. Tests have shown that it takes more than $\sqrt{\pi q/2}$ iterations for f_P to reach a collision [23, 24], so that f_P is not an optimal choice for an iterating function.

***r*-adding walks** Let $3 \leq r \leq 100$ be a small positive integer and let $G = T_0 \cup \cdots \cup T_{r-1}$ be a partition of G into r-many subsets of roughly the same size. The index function $s : G \to \{0, 1, \ldots, r-1\}$ is defined by setting $s(y) = s$ for $y \in T_s$. For each $s = 0, \ldots, r-1$, randomly choose integers $m_s, n_s \in \mathbf{Z}/q\mathbf{Z}$ and set the multipliers to $M_s = g^{m_s} h^{n_s}$. The iterating function is given by

$$f_T(y) = yM_{s(y)}.$$

That is, one of the r-many fixed elements $M_s \in G$ is multiplied, depending on which subset T_s the input belongs to. This is clearly exponent traceable, with the exponent functions being addition by m_s and n_s. The name r-adding refers to the additions.

This method was introduced in [17] and the work [16] shows that any $r \geq 8$ will suffice for cyclic groups. Testing [24] on cyclic elliptic curve groups show that 20-adding walks perform very close to a random function.

2.3 Collision detection

The main issues with collision detection is to detect a collision with minimal number of additional iterating function applications after collision occurs, and with a small amount of memory. There have been several proposals on collision detection methods by Floyd [9], Brent [2], Sedgewick-Szymanski-Yao [19], Quisquater-Delescaille [15], and Nivasch [11].

Among them, the method using distinguished points by Quisquater and Delescaille [15] is regarded as the most efficient one. This was originally an idea

for use with time-memory trade-off techniques. Distinguished points are those elements of G that satisfy a certain condition, which is easy to check. For example, with a fixed encoding for G, we may set them to be those elements with a certain number of starting bits equal to zero.

After each application of the iterating function, the current g_i is stored in a table, if it is a distinguished point. The algorithm terminates when a collision is found among the distinguished points. The distinguished points should be defined so that this table is of manageable size.

Let θ be the fraction of elements in G which satisfy the distinguishing property. The algorithm is expected to terminate with a collision after $\sqrt{\pi q/2} + 1/\theta$ applications of the iterating function.

This method has the advantage that it can lead to n-times speedup with n-processor parallelization [12].

3 Tag Tracing

Let us recall the r-adding walk iterating function f_T. Given an input $g_i \in G$, it first determines the index $s = s(g_i)$, and produces $g_{i+1} = g_i M_s \in G$ as the output. Occasionally, the output $g_i M_s$ is placed in a table of small size.

Notice that the storing operation is not very frequent. So, one may question whether computing the product $g_i M_s$ is really necessary at every iteration. Of course, iterated applications of f_T require current g_i to be available, but this is avoidable if we have a pre-computed table of suitably many products of M_s. Then it suffices for one to compute just the index at each iteration. We shall explore this line of reasoning in this section.

3.1 Preparation

As in the r-adding walk, we fix an index set $\mathcal{S} = \{0, 1, \ldots, r-1\}$ for some small r and let $\mathcal{M} = \{M_s = g^{m_s} h^{n_s}\}_{s \in \mathcal{S}}$ be a multiplier set for the r-adding walk. Fix a small positive number ℓ and consider the product set $\mathcal{M}_\ell = (\mathcal{M} \cup \{1\})^\ell$, i.e., the set of products of at most ℓ-many M_s. Notice that we know how to write each element of $\mathcal{M}_\ell$ in the form $g^m h^n$. We shall treat the set $\mathcal{M}_\ell$ as a table of elements of G, listed together with their respective exponent forms.

For our tag tracing approach to the DLP, we want to pre-compute $\mathcal{M}_\ell$ before going into the actual r-adding walk, and the following two lemmas show the range of r and ℓ one may choose, depending on the resources available.

Lemma 1. *The size of $\mathcal{M}_\ell$ is at most $\binom{\ell+r}{r}$.*

Proof. The size of $\mathcal{M}_\ell$ is bounded above by the number of combinations with repetitions, where one chooses ℓ times from the set $\mathcal{M} \cup \{1\}$ of size $r+1$. The bound is reached only if all product elements produced are distinct. □

Lemma 2. *The set $\mathcal{M}_\ell$ can be constructed in $\binom{\ell+r}{r} - 1$ multiplications in G.*

Proof. Consider the complete r-ary tree structure of depth ℓ. Label each edge with an index from $\mathcal{S}$ in such a way that from each node, the r edges extending to its children nodes are labeled with different indices. We label the root node with $1 \in G$ and label each node below with the element of $\mathcal{M}_\ell$ which is the product of multipliers labeled by edges on its way down.

The nodes of the complete r-ary tree will contain multiple copies of $\mathcal{M}_\ell$. It is clear that if we collect just the nodes with paths to the root that are labeled in non-decreasing order, then we will obtain one copy of $\mathcal{M}_\ell$. As the number of edges leading to these nodes is one less than the number of these nodes, and since each edge corresponds to one multiplication used in creation of node labels, we arrive at our claim. □

Some example values would be $\binom{84}{20} \sim 2^{63.2}$ for 20-adding walks with $\ell = 64$, and $\binom{72}{8} \sim 2^{33.4}$ for 8-adding walks with $\ell = 64$. As $\mathcal{M}_\ell$ can only be computed after h, whose discrete logarithm we are looking for, is given, we do not want these pre-computation complexities to go over our main attack complexity.

We now fix a *tag set* $\mathcal{T}$ together with three functions.

$$\tau : G \to \mathcal{T}.$$
$$\bar{\tau} : G \times \mathcal{M}_\ell \to \mathcal{T} \cup \{\text{fail}\}.$$
$$\sigma : \mathcal{T} \to \mathcal{S} = \{0, 1, \ldots, r-1\}.$$

The first function τ is named the *tag function*. We define the index function $s : G \to \mathcal{S}$ to be $s = \sigma \circ \tau$ and also consider the function $\bar{s} = \sigma \circ \bar{\tau} : G \times \mathcal{M}_\ell \to \mathcal{S} \cup \{\text{fail}\}$. The three functions above are to be chosen so that they satisfy the following condition.

1. The index function $s = \sigma \circ \tau$ is surjective and roughly pre-image uniform, i.e., grouping G according to its image points under s partitions G into subsets of roughly the same size.
2. When $\bar{s}(g, M) \in \mathcal{S}$, we have $\bar{s}(g, M) = s(g \cdot M)$. In particular, any successful output of $\bar{s}$ depends only on the product of its inputs.

So we are looking for a function τ that resembles a normal index function, but with a larger image set, and also another way $\bar{\tau}$ to evaluate τ on product of group elements.

The situation we have in mind concerning τ and $\bar{\tau}$ is as follows. Given a random $M \in \mathcal{M}_\ell$ and $g \in G$, the expected time for calculation of $\bar{\tau}(g, M)$ is smaller than the time needed for computation of the product $M \cdot g$. The general thought behind this is that it should take less effort to obtain some partial information about a product than the full product itself. For example, consider the case $G \subset \mathbf{F}_p^\times$ and define $\tau(g)$ to be the most significant k bits of $g \in G$. Intuitively, computing k bits out of the $\log p$ bits of product gM may take as little as $\frac{k}{\log p}$ of the time for full product computation. If some of the product bits were easier to calculate than others, the time could be even shorter.

We shall denote the expected time for $\bar{s}(g, M)$ evaluation by $|\bar{s}|$.

3.2 Iterating function

The iterating function of our tag tracing algorithm will follow the usual r-adding walks. We have already fixed an index set $\mathcal{S} = \{0, 1, \ldots, r-1\}$ with an appropriate index function $s = \sigma \circ \tau$ and a multiplier set $\mathcal{M}$ during the preparation phase.

We start with a random $g_0 \in G$ and the first index $s_0 = s(g_0)$ is computed. We set $g_1 = g_0 M_{s_0}$, exactly as in the normal r-adding walk process, but the product $g_0 M_{s_0}$ is *not* computed. Instead, $\bar{s}(g_0, M_{s_0})$ is computed in time $|\bar{s}|$. If $\bar{s}(g_0, M_{s_0}) \in \mathcal{S}$, we have computed

$$s_1 = s(g_1) = s(g_0 M_{s_0}) = \bar{s}(g_0, M_{s_0}).$$

We have not fully computed g_1, but can set $g_2 = g_1 M_{s_1} = g_0 M_{s_0} M_{s_1}$, which is, once again, not computed.

Now, since $M_{s_0} M_{s_1} \in \mathcal{M}_\ell$ is an element which has been pre-computed, we can evaluate $\bar{s}(g_0, M_{s_0} M_{s_1})$ in time $|\bar{s}|$. This leads us to index value s_2 and we can continue as before.

If we come across the situation $\bar{s}(g_0, M_{s_0} \cdots M_{s_k}) \notin \mathcal{S}$, or arrive at ℓ iterations of the above process, we do a full product computation. That is, we compute $g_{k+1} = g_0 M_{s_0} \cdots M_{s_k}$ and let this replace the role g_0 has taken up to that iteration. Notice that this full product requires just one multiplication, since $M_{s_0} \cdots M_{s_k} \in \mathcal{M}_\ell$ has been pre-computed.

Notice that since the set $\mathcal{M}_\ell$ is a table of elements of G, listed together with its respective exponent forms, the above process is fully exponent traceable.

3.3 Collision detection

To complete the description of the tag tracing method, we need to check if is possible to detect collisions. The distinguished point method is well suited for our tag tracing.

Usually, the distinguished points is defined to be points with a certain number of starting bits equal to zero, under a fixed encoding. With tag tracing, we use this usual definition, but for more efficiency, impose an additional condition to be satisfied. This extra condition is set to depend on the tag value $\tau(g)$ in such a way that it can only be satisfied when $\bar{\tau}(g', M') \notin \mathcal{T}$ for every g' and M' such that $g = g'M'$. Then, whenever there is a chance of some g_i being a distinguished point, we would already have the full form for g_i, and there are no additional full product computations involved in relation to collision detection. With the extra condition on the tag, the original condition can be relaxed to maintain the number of distinguished points.

3.4 Complexity analysis

Let us make a rough time complexity comparison of our tag tracing with the original r-adding walks. The storage complexity of tag tracing is given by Lemma 1.

We can assume that the various parameters for tag tracing has been chosen so that the time taken for preparation, given by Lemma 2, is insignificant compared to the main function iterations. We shall also not include efforts needed in following through the exponents needed in final computation of the discrete logarithm.

Consider the time taken to do a full product computation in G. This will be almost equal to $|f_T|$, the time taken for one iteration of the r-adding walk. Even though this time will depend on the encoding for G, we shall assume that computation of full product in our tag tracing also requires time $|f_T|$.

Recalling the notation $|\bar{s}|$ introduced earlier, we can restate one of our requirements on $\bar{\tau}$ as $\frac{|\bar{s}|}{|f_T|} < 1$. It is now easy to see that a single iteration of tag tracing is expected to take time

$$|\bar{s}| + \big(\frac{1}{\ell} + P_{\text{fail}}\big)|f_T|, \tag{1}$$

at the most, where P_{fail} is the probability of reaching $\bar{s}$ value not in $\mathcal{S}$.

The expected running time of tag tracing is the above value multiplied by the number of iterations required for a normal r-adding walk style algorithm. Hence the ratio of running time between tag tracing and a normal r-adding walk would be

$$\frac{|\bar{s}|}{|f_T|} + \frac{1}{\ell} + P_{\text{fail}}.$$

If this is less than 1, we have a reduction in discrete logarithm solving time. As discussed earlier, it should be possible to find τ and $\bar{\tau}$ such that $|\bar{s}|$ is much smaller than $|f_T|$, making the above a meaningful reduction in time.

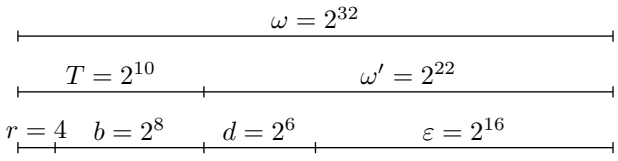
4 Application to Prime Fields and its Implementation

Throughout this section, p will be a prime and $G = \langle g \rangle \subset \mathbf{F}_p^{\times}$ will be a cyclic group of order q. We will show how to apply the proposed tag tracing algorithm to G and present some implementation results. Tag tracing on subgroups of the binary field, which is quite similar, is dealt in Appendix B.

4.1 Parameter setup

We fix the index set size r and the multiplier product pre-computation length ℓ in such a way that the time and storage complexities given by Lemma 1 and Lemma 2 are manageable. The tag set $\mathcal{T} = \{0, 1, 2, \ldots, T-1\}$ is taken to be of size $T = r \cdot b$, a multiple of r. We take a positive integer ε and set $d = \lceil \log_{\varepsilon} p \rceil$. Then we choose integer $\omega' \geq d(\varepsilon - 1) + 1$. We use the notation $\omega = T\omega'$ and assume that $\omega < p^{\frac{1}{3}}$.

Optimal choice for these parameters will depend on many factor including the size of prime p, resources available, and the speed of large integer multiplications. The parameter set below with $\ell = 128$ may be appropriate for use on a modern PC when primes p is of 1024-bit size. Readers may keep these in mind to facilitate understanding of further material.



4.2 Tag function

Our assumption $\omega < p^{\frac{1}{3}}$ implies that we may always choose integer $B > p^{\frac{2}{3}}$ such that $0 \leq \omega B - p < B^{\frac{1}{2}}$. For example, setting $B = \lceil p/\omega \rceil$ should always work. We fix any such B and define the tag function $\tau : G \to \mathcal{T}$ as

$$\tau(g) = \left\lfloor \frac{g \bmod p}{\omega' B} \right\rfloor, \tag{2}$$

where we are using "$x \bmod y$" to denote the unique integer between 0 and $y-1$ that is congruent to x modulo y. Notice that $0 \leq \omega B - p$ implies $\frac{p-1}{\omega' B} < T$, so that the above quotient indeed lies in $\mathcal{T} = \{0, 1, \ldots, T-1\}$. We also define $\sigma : \mathcal{T} \to \mathcal{S} = \{0, 1, \ldots, r-1\}$ as $\sigma(x) = \lfloor x/b \rfloor$ and this fixes the index function $s = \sigma \circ \tau : G \to \mathcal{S}$.

The following lemma shows that we can expect τ to be roughly pre-image uniform.

Lemma 3. *If variable* $\mathbf{x}$ *is uniformly distributed over* $\mathbf{F}_p$*, then the probability distribution of* $\tau(\mathbf{x})$ *over* $\mathcal{T}$ *is almost uniform in the sense that*

$$\left|\mathrm{Prob}[\tau(\mathbf{x}) = k] - \mathrm{Prob}[\tau(\mathbf{x}) = k']\right| < \frac{1}{p^{\frac{1}{2}}}$$

for any $k, k' \in \mathcal{T}$*.*

Proof. We view τ as having been defined on all of $\mathbf{F}_p$. Note that $\bar{p} := T\omega' B - p = \omega B - p < B^{\frac{1}{2}} < \omega' B$. This implies that for each fixed $k = 0, \ldots, T-2$, there are exactly $\omega' B$ elements $0 \leq \mathbf{x} < p$ with $\tau(\mathbf{x}) = k$ and that there are $\omega' B - \bar{p}$ elements satisfying $\tau(\mathbf{x}) = T-1$. Thus the maximal difference between pre-image sizes is $\bar{p}$. Notice that the condition $\omega B - p < B^{\frac{1}{2}}$ implies $B < p$. The maximal probability difference can now be seen to be less than $\bar{p}/p < B^{\frac{1}{2}}/p < p^{-\frac{1}{2}}$. □

Since the condition $T = r \cdot b$ makes σ exactly pre-image uniform, the above lemma holds even when $\tau(\mathbf{x})$ is replaced by $s(\mathbf{x})$, and we can state the following.

Proposition 1. *Assuming that the elements of* G *are uniformly distributed over* $\mathbf{F}_p$*, we can expect the index function* s *to be roughly pre-image uniform.*

4.3 Auxiliary functions

We should now present the auxiliary function $\bar{\tau} : G \times \mathcal{M}_\ell \to \mathcal{T} \cup \{\text{fail}\}$ which is essentially equal to τ.

Given $\mathbf{x}, \mathbf{y} \in \mathbf{F}_p$, we can always write

$$\mathbf{x} = \sum_{i=0}^{d-1} x_i \varepsilon^i \quad (0 \le x_i < \varepsilon)$$

and, for each $0 \le i \le d-1$, we can write

$$\varepsilon^i \mathbf{y} \bmod p = \hat{y}_i B + \check{y}_i \quad (0 \le \hat{y}_i \le \frac{p-1}{B} < \omega,\, 0 \le \check{y}_i < B). \tag{3}$$

Using this notation, we define

$$\bar{\bar{\tau}}(\mathbf{x}, \mathbf{y}) = \left\lfloor \frac{\sum_{i=0}^{d-1} x_i \hat{y}_i \bmod \omega}{\omega'} \right\rfloor. \tag{4}$$

Let us check how close $\bar{\bar{\tau}}(\mathbf{x}, \mathbf{y})$ is to $\tau(\mathbf{xy})$.

Lemma 4. *Given $\mathbf{x}, \mathbf{y} \in \mathbf{F}_p$, we have $\tau(\mathbf{xy}) = \bar{\bar{\tau}}(\mathbf{x}, \mathbf{y})$ or $\bar{\bar{\tau}}(\mathbf{x}, \mathbf{y}) + 1$, unless $\bar{\bar{\tau}}(\mathbf{x}, \mathbf{y}) = T - 1$.*

Proof. Before going into the proof, for easy reference, let us recall some of the conditions that were placed on the parameters: $d(\varepsilon - 1) < \omega'$; $\omega = T\omega'$; $\omega < p^{\frac{1}{3}} < B^{\frac{1}{2}}$; $\omega B - p < B^{\frac{1}{2}}$;

We start by writing

$$\sum_{i=0}^{d-1} x_i \hat{y}_i = a_2 \omega + a_1 \omega' + a_0,$$

where the coefficients a_0, a_1, and a_2 are to be obtained through usual integer divisions. In particular, we have $a_2 \le \frac{d(\varepsilon-1)(\omega-1)}{\omega} < d(\varepsilon - 1) < \omega' < \omega < B^{\frac{1}{2}}$. It should also be noted that $a_1 = \bar{\bar{\tau}}(\mathbf{x}, \mathbf{y})$.

In the above notation, we may write

$$\begin{aligned} \mathbf{xy} = \sum_{i=0}^{d-1} x_i \varepsilon^i \mathbf{y} &\equiv \Big(\sum_{i=0}^{d-1} x_i \hat{y}_i \Big) B + \sum_{i=0}^{d-1} x_i \check{y}_i \pmod p \\ &\equiv a_1 \omega' B + a_0 B + a_2(\omega B - p) + \sum_{i=0}^{d-1} x_i \check{y}_i \pmod p. \end{aligned}$$

The various conditions allow us to bound the lower terms by

$$\begin{aligned} a_0 B + a_2(\omega B - p) + \sum_{i=0}^{d-1} x_i \check{y}_i & \\ &< (\omega' - 1)B + B^{\frac{1}{2}} \cdot B^{\frac{1}{2}} + d(\varepsilon - 1)(B - 1) \\ &< \omega' B + (\omega' - 1)B = 2\omega' B - B. \end{aligned}$$

Now, if $a_1 = \bar{\bar{\tau}}(\mathbf{x}, \mathbf{y})$ is strictly less than $T-1$, then

$$\begin{aligned} a_1\omega' B + a_0 B + a_2(\omega B - p) + \sum_{i=0}^{d-1} x_i \check{y}_i \\ < (T-2)\omega' B + 2\omega' B - B = \omega B - B < p. \end{aligned}$$

So, when $\bar{\bar{\tau}}(\mathbf{x}, \mathbf{y}) \neq T-1$, we know

$$\mathbf{xy} \bmod p = a_1\omega' B + \Big\{ a_0 B + a_2(\omega B - p) + \sum_{i=0}^{d-1} x_i \check{y}_i \Big\}.$$

Finally, since the sum of terms inside the braces is non-negative and strictly less than $2\omega' B$, the quotient of $\mathbf{xy} \bmod p$ divided by $\omega' B$ must be either a_1 or $a_1 + 1$. □

Based on this lemma, we define $\bar{\tau} : G \times \mathcal{M}_\ell \to \mathcal{T} \cup \{\text{fail}\}$ as follows.

$$\bar{\tau}(g, M) = \begin{cases} \text{fail} & \text{if } \bar{\bar{\tau}}(g, M) \bmod b \text{ is either } b-1 \text{ or } b-2, \\ \bar{\bar{\tau}}(g, M) & \text{if otherwise.} \end{cases}$$

Recalling the definitions $\sigma(x) = \lfloor x/b \rfloor$, $s = \sigma \circ \tau$, and $\bar{s} = \sigma \circ \bar{\tau}$, it is now easy to show the following proposition.

Proposition 2. *When $\bar{\tau}(g, M) \in \mathcal{T}$ and hence $\bar{s}(g, M) \in \mathcal{S}$, we have $s(g \cdot M) = \bar{s}(g, M)$ and $\tau(gM) \bmod b \neq b-1$.*

Proof. We note that $T - 1 \bmod b = b - 1$, so that Lemma 4 together with $\bar{\bar{\tau}}(g, M) \bmod b \neq b-1$ implies $\tau(gM) = \bar{\bar{\tau}}(g, M)$ or $\bar{\bar{\tau}}(g, M) + 1$.

Now, this together with the condition that $\bar{\bar{\tau}}(g, M) \bmod b$ is neither $b-1$ nor $b-2$ implies $\tau(gM) \bmod b \neq b-1$.

In addition, we have $\bar{\tau}(g, M) = \bar{\bar{\tau}}(g, M$ and $\bar{\tau}(g, M) \bmod b \neq b-1$ implies

$$\lfloor \bar{\tau}(g, M)/b \rfloor = \lfloor (\bar{\tau}(g, M) + 1)/b \rfloor,$$

which must be $s(g \cdot M)$. □

4.4 Tag tracing

We are now ready to start tag tracing. Using the proof of Lemma 2 as a hint, we compute a table containing entries (M, m, n) for $M = g^m h^n \in \mathcal{M}_\ell$. We also append the associated vector

$$\mathbf{v}_{\varepsilon,B}(M) = \left(\left\lfloor \frac{\varepsilon^0 M \bmod p}{B} \right\rfloor, \ldots, \left\lfloor \frac{\varepsilon^{d-1} M \bmod p}{B} \right\rfloor \right)$$

to each entry of the table. Notice that these are the $\hat{y}_i$ appearing in equation (3).

We can now follow the discussion of Section 3.2 to compute each iteration of tag tracing. The elements of G are written in ε-ary representation so that we may quickly compute s using equation (4) and Proposition 2. Whenever we reach ℓ iterations or an $\bar{s}$ calculation failure, the complete product g_i is computed using one group multiplication. A point $g \in G$ is defined to be a distinguished points only if $\tau(g) \bmod b = b - 1$ and if it satisfies some additional on g. According to Proposition 2, $g \cdot M$ can be a distinguished point only when $\bar{\tau}(g, M) \notin \mathcal{T}$.

4.5 Implementation

We have tested tag tracing with an implementation on a modern PC and compared it with a normal 20-adding walk. Both the tag tracing and 20-adding walks were set to use distinguished points for collision detection. We used the finite field arithmetics provided by the NTL [21] library to implement the 20-adding walk, so as not to be biased. Throughout the test, prime p was taken to be of 1024-bit size, and whenever random primes p, q and $\langle g \rangle \subset \mathbf{F}_p^\times$ of order q was needed, they were generated in the style specified for DSA [5].

After comparing rho lengths of r-adding walks for various r, we opted to use $r = 4$ for tag tracing, as we did not have much memory available. Compared to the 20-adding walk, our tag tracing with $r = 4$ will have approximately 1.3 times longer rho length. This is explained in Appendix A. Other parameters were set to $b = 2^8$, $\varepsilon = 2^{16}$, $T = 2^{10}$, $\omega' = 2^{22}$, and $\omega = 2^{32}$.

For speed comparison, we chose q to be of 160-bit size and ran both the 20-adding walk and tag tracing for 2^{28} iterations. For tag tracing this was done for various choices of ℓ, with a set of randomly chosen primes p and q, group generator g, DLP target h, adding walk multipliers M_s, and initial starting point. Timings are listed in Table 1. The size $\binom{\ell+r}{r} \cdot \big((1 + \|\omega\|/\|\varepsilon\|)\|p\| + 2\|q\|\big)$ of table $\mathcal{M}_\ell$ and its preparation time is also listed, where the $\|\cdot\|$ notation has been used for bit length. The corresponding time, averaged over 10 randomly generated starting points was 1071.4 seconds for the 20-adding walk.

ℓ	10	20	30	40	50	60	70	80	90	100
$\lvert\bar{s}\rvert$ (sec)	156.6	91.8	75.0	70.5	70.3	70.0	70.8	71.6	72.6	73.9
$\lvert f_T\rvert/\lvert\bar{s}\rvert$	6.8	11.7	14.3	**15.2**	**15.2**	**15.3**	15.1	15.0	14.8	14.5
$\mathcal{M}_\ell$ size (MB)	0.4	4.3	18.8	54.9	127.9	256.9	465.3	780.2	1233.1	1859.3
$\mathcal{M}_\ell$ comp time (sec)	0.21	2.27	9.90	29.1	68.0	137	245	414	650	983

Table 1. Tag tracing timing for 2^{28} iterations ($\|q\| = 160$)

The table shows that the speed of tag tracing iteration can be over 15 times faster than that of a 20-adding walk. Since the rho length of a 4-adding walk is 1.3 times longer than that of a 20-adding walk, this translates to tag tracing being more than 11.5 times faster than a 20-adding walk in solving DLP.

Table 1 is also interesting in that it reflects the complexity estimate given by equation (1). Larger ℓ imply smaller number of full product computation and this results in steep increased speed for $\ell = 10 \sim 60$. The gradual decrease in speed after that seems to be from two factors. As we are using $b = 2^8$, we have $P_{\text{fail}} = 1/2^7$, and the increasing of ℓ looses effect as we approach $\ell = 2^7$. We have experienced through various tweaks that table lookups to $\mathcal{M}_\ell$ present a considerable fraction of the time taken by a tag tracing iteration. This coupled with our poor use of memory is another reason for decrease in speed at high ℓ. In any case, unlike our primitive testing, large scale implementation of tag tracing will need to use advanced hash table techniques that allow constant time table lookups.

We verified with small q that tag tracing has no problem in solving DLPs. Except for the q size, parameters identical to the above were used with $\ell = 40$. The timings, averaged over 200 randomly generated starting points and multiplier sets, are given in Table 2. The figures do not include the approximately 29 seconds spent on creation of $\mathcal{M}_\ell$. This may seem illogical here, but as table creation time does not change much with q, the speed ratio calculated in this way will reflect what can be expected of the ratio at large q. The data in Table 2 roughly coincides with our prediction of 11.5 factor speedup.

	$\|q\| = 35$	$\|q\| = 40$	$\|q\| = 45$	$\|q\| = 50$	$\|q\| = 55$
Pollard rho	1.103 sec	6.272 sec	38.738 sec	203.138 sec	1185.578 sec
20-adding	0.940 sec	5.174 sec	29.653 sec	159.977 sec	959.027 sec
tag tracing	0.093 sec	0.441 sec	2.634 sec	13.481 sec	80.785 sec
Pollard rho/tag tracing	11.89	14.24	14.70	15.07	14.68
20-adding/tag tracing	10.14	11.75	11.26	11.87	11.87

Table 2. Full running time comparison of tag tracing and 20-adding walks

5 Asymptotic Complexity

In this section, we consider the asymptotic complexity of the proposed algorithm for large p. We will use $\mathrm{Mul}(k)$ to denote the cost of multiplication modulo an integer of k-bit size.

Looking at equation (4) and the definition of σ, we can check that the cost of evaluating the auxiliary index function $\bar{s}$ is d multiplications modulo ω, $d-1$ additions modulo ω, and two divisions of integers less than ω. Thus, ignoring the small fixed number of divisions and the relatively cheaper additions, we can say that $\bar{s}$ evaluation costs approximately $d\,\mathrm{Mul}(\|\omega\|)$. Recalling (1), we can write the average cost of a single tag tracing iteration as

$$d\,\mathrm{Mul}(\|\omega\|) + \left(\frac{1}{\ell} + \frac{2}{b}\right)\mathrm{Mul}(\|p\|).$$

If ω is set to grow with p, this complexity would not be linear in $k = \|p\|$. To obtain linear complexity, we perform tag computation in an incremental way, starting with fixed small parameters and recomputing with incrementally larger parameters only when the previous attempt fails. Let us explain the procedure in more detail.

We fix $r \geq 4$ and $b \geq 2$ to be small constants and define $t = \lfloor \log_b k \rfloor$. We take $\ell = \Theta(k)$, fix ε to a positive integer satisfying $\varepsilon^t \leq p^{\frac{1}{3}}/(rk^2)$, and let $d = \lceil \log_\varepsilon p \rceil$, as before. Based on these, we prepare a parameter set for each index $i = 1, \ldots, t$, as follows: $b_i = b^i$, $T_i = rb_i$, $\varepsilon_i = \varepsilon^i$, $\omega'_i = d\varepsilon_i$, $\omega_i = T_i\omega'_i$, $B_t = \lceil p/\omega_t \rceil$, $B_i = \left(\frac{\omega_t}{\omega_i}\right) B_t$.

Note that ω'_i does not involve $d_i = \lceil \log_{\varepsilon_i} p \rceil$, allowing each ω_i to divide ω_{i+1} and making each B_i an integer. It is possible to check that each set of parameters satisfy all conditions set forth on Section 4.1 and Section 4.2. For example, $\omega_i \leq \omega_t = rb^t d\varepsilon^t < p^{\frac{1}{3}}$ and $\omega'_i = d\varepsilon_i \geq d_i\varepsilon_i \geq d_i(\varepsilon_i - 1) + 1$. We can also use $p^{\frac{2}{3}} < p/\omega_t \leq B_t$ to show $0 \leq \omega_i B_i - p = \omega_t B_t - p < \omega_t < p^{\frac{1}{3}} < B_t^{1/2} \leq B_i^{\frac{1}{2}}$.

For each i, we can define the tag function τ_i and the index function s_i as in Section 4, i.e.,

$$\tau_i(g) = \left\lfloor \frac{g \bmod p}{\omega'_i B_i} \right\rfloor, \quad s_i(g) = \left\lfloor \frac{\tau_i(g)}{b_i} \right\rfloor = \left\lfloor \frac{g \bmod p}{\omega'_i B_i b_i} \right\rfloor. \tag{5}$$

Since $b_i\omega'_i B_i = b_t\omega'_t B_t$, we have $s_i(g) = s_t(g)$, for any i. We already know that this common index function is roughly pre-image uniform.

Let $g \in G$ and $M \in \mathcal{M}_\ell$. For each i, we can define $\bar{\tau}_i(g, M)$ as in Section 4, which is computed in time $d_i \operatorname{Mul}(\|\omega_i\|)$, and is successful in giving $s_i(g \cdot M)$ with probability $1 - 2/b^i$. We now use an incremental approach in computing the common index $s_1(g \cdot M)$. First, $\bar{\tau}_1(g \cdot M)$ is computed. If it returns a failure, we compute $\bar{\tau}_2(g \cdot M)$, and so on. We stop whenever an output $s_i(g \cdot M)$ for $i \leq t$ is successfully obtained and move onto the next iteration of tag tracing. The full product of g and M is computed if all t attempts fail.

Then the time complexity of this incremental approach is

$$d_1 \operatorname{Mul}(\|\omega_1\|) + \frac{2d_2}{b} \operatorname{Mul}(\|\omega_2\|) + \cdots + \frac{2d_t}{b^{t-1}} \operatorname{Mul}(\|\omega_t\|) + \left(\frac{1}{\ell} + \frac{2}{b^t}\right) \operatorname{Mul}(\|p\|)$$
$$\leq 2\left(\frac{b}{b-1}\right)^2 \operatorname{Mul}(\|\omega_1\|) \lceil \log_\varepsilon p \rceil + \left(\frac{1}{\ell} + \frac{2}{b^t}\right) \operatorname{Mul}(\|p\|) = O(\|p\|) = O(k),$$

where we have used the facts $\omega_i < \omega^i$, $d_i \leq \lfloor d/i \rfloor$, $\ell = \Theta(k)$, $b^t = \Theta(k)$, and that $\operatorname{Mul}(k)$ is at most quadratic in k.

The incremental approach requires t tables and since an entry in the i-th table is of $d_i\|\omega_i\| < \log_\varepsilon \omega \log p$ bits, noting that $\prod_{i=1}^{r} \frac{\ell+i}{i} \leq \ell^r = O(k^r)$, we can write the storage requirement as

$$t\binom{\ell + r}{r} (\log_\varepsilon \omega \cdot \log p) = O(k^{r+1} \cdot \log k).$$

It only remains to consider collision detections. A point $g \in G$ is defined to be a distinguished point only if $\tau_t(g) \bmod b^t = b^t - 1$ with possibly some additional

conditions on g. Because $\lfloor \tau_i(g)/b^i \rfloor = \lfloor \tau_t(g)/b^t \rfloor$ and $\tau_i(g) = \lfloor \tau_t(g)/b^{t-i} \rfloor$, we have $\tau_t(g) \bmod b^t = \Big(\tau_i(g) \bmod b^i\Big) b^{t-i} + \tau_t(g) \bmod b^{t-i}$. From this, we see that $\tau_t(g) \bmod b^t = b^t - 1$ implies $\tau_i(g) \bmod b^i = b^i - 1$ for any i. Thus distinguished point candidates can be noticed from any $\bar{\tau}_i(g, M)$.

6 Conclusion

In this paper, we proposed a method to speed up the Pollard rho algorithm on cyclic subgroups of the prime field $\mathbf{F}_p$. The proposed algorithm replaces the multiplication needed in r-adding walks with an operation of linear complexity. As a further work, we would like to generalize our algorithms to elliptic or hyperellipic curves.

Acknowledgments. This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant (No. R01-2008-000-11287-0).

References

1. L. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," Proc. of the IEEE 20th Annual Symposium on Foundations of Computer Science (FOCS), pp.55-60, 1979.
2. R. Brent, "An improved Monte Carlo Factorization Algorithm," *BIT*, Vol. 20, pp. 176-184, 1980.
3. W. Diffie and M. Hellman, "New Directions in Cryptology," *IEEE Trans. Inform. Theory*, Vol. 22, pp. 644-654, 1976.
4. I. Duursma, P. Gaudry, and F. Morain, "Speeding up the Discrete Log Computation on Curves with Automorphisms," *Asiacrypt '99*, LNCS 1716, Springer-Verlag, pp. 103-121, 1999.
5. "Digital Signature Standard," NIST. U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 186, May 1994.
6. T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms," *IEEE Trans. Infrom. Theory*, Vol. 31, pp. 469-472, 1985.
7. R. Gallant, R. Lambert, and S. Vanstone, "Improving the Parallelized Pollard Lambda Search on Binary Anomalous Curves," *Math. Comp.*, Vol. 69, pp. 1699-1705, 2000.
8. A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics-Doklady*, Vol. 7, pp.595-596, 1963.
9. D. Knuth, *The Art of Computer Programming, vol. II: Seminumerical Algorithms*, Addison-Wesley, 1969.
10. D. Knuth, *The Art of Computer Programming, vol. III: Sorting and Searching*, Addison-Wesley, 1973.
11. G. Nivasch, "Cycle Detection using a Stack," *Information Processing Letters*, Vol. 90, pp. 135-140, 2004.
12. P. van Oorschot and M. Wiener, "Parallel Collision Search with Cryptanalytic Applications," *J. Cryptology*, Vol. 12, pp. 1-28, 1999.

13. S. Pohlig and M. Hellman, "An Improved Algorithm for Computing Discrete Logarithms over $GF(p)$ and its Cryptographic Significance," *IEEE Trans. Inform. Theory*, Vol. 24, pp. 106-110, 1978.
14. J. Pollard, "A Monte Carlo Method for Index Computation (*modp*)," *Math. Comp*, Vol. 32(143), pp. 918-924, 1978.
15. J. Quisquater and J. Delescaille, "How easy is Collision Search? Application to DES," Proc. of Eurocrypt, Springer-Verlag, LNCS 434, pp. 429-434, 1989.
16. J. Sattler and C. Schnorr, "Generating Random Walks in Groups," *Ann.-Univ.-Sci.-Budapest.-Sect.-Comput.*, Vol. 6, pp.65-79, 1985.
17. C. Schnorr and H. Lenstra, Jr., "A Monte Carlo Factoring Algorithm with Linear Storage," *Math Comp.*, Vol. 43(167), pp. 289-311, 1984.
18. A. Schönhage and V. Strassen, "Schnelle Multiplikation Grobner Zahlen," *Computing*, Vol. 7, pp.281-292, 1971.
19. R. Sedgewick, T. Szymanski, and A. Yao, "The Complexity of Finding Cycles in Periodic Functions," *SIAM Journal on Computing*, Vol. 11, No. 2, pp. 376-390, 1982.
20. D. Shanks, "Class number, a Theory of Factorization and Genera," Proc. Symp. Pure Math., Vol. 20, pp. 415-440, 1971.
21. V. Shoup, NTL: A Library for doing Number Theory, Ver 5.4.1, http://shoup.net/ntl/.
22. V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, 2005.
23. E. Teske, "Speeding up Pollard's rho Method for Computing Discrete Logarithms," *ANTS* III, Springer, LNCS 1423, pp. 541-554, 1998.
24. E. Teske, "On Random Walks for Pollard's rho Method," *Math. Comp.*, Vol. 70, pp. 809-825, 2001.
25. M. Wiener and R. Zuccherato, "Fast Attacks on Elliptic Curve Cryptosystems," *Selected Areas in Cryptography '98*, LNCS 1556, Springer-Verlag, pp.190-200, 1999.

A Performance of 4-adding Walks

For $r \geq 3$, let us write f_r to denotes the r-adding walk iterating function. We also write f_P for the Pollard's iterating function. Where as the rho length of a function graph on a set of size q is expected to be $\sqrt{\pi q/2}$ for a random function, the actual rho lengths of various f_r and f_P are a small constant multiple of $\sqrt{\pi q/2}$. We shall write C_r and C_P for these constants. In this section, we show experiment results on these values. During the test, size of p was always set to 1024 bits, but varying q sizes were used.

In order to use the iterating functions f_r and f_P we need to define an index function. For each $r \geq 3$, the index function $s_r : \mathbf{F}_p \rightarrow \{0, \ldots, r-1\}$ was set to $s_r(g) = \lfloor r \cdot (A \cdot g \mod 1) \rfloor$, where A is a rational approximation of the golden ratio $(\sqrt{5}-1)/2$. When A is of sufficient precision, this is known to bring about uniform looking distribution [10], even on non-uniform inputs. For our experiment, a precision of 1044 binary places for A is sufficient.

Estimates for the constants C_r and C_P were found as follows. Primes p, q and cyclic group generator g of order q in $\mathbf{F}_p^{\times}$ were randomly generated in the DSA style [5], and the multiplier set was randomly selected. Then the iterating

function was iterated from a random starting point until the walk intersected itself in a rho. The length of the rho was recorded and the process redone with a newly generated g, h and multiplier set. This was repeated 1000 times for each iterating function. The average rho lengths divided by $\sqrt{\pi q/2}$ are the constant

$\|q\|$	10	15	20	25	30	35	40
C_P	1.244	1.267	1.307	1.289	1.304	1.325	1.312
C_3	1.628	1.830	2.051	2.201	2.408	2.568	2.742
C_4	1.336	1.346	1.328	1.374	1.360	1.368	1.370
C_8	1.092	1.105	1.072	1.087	1.061	1.098	1.058
C_{20}	0.995	1.008	1.036	1.004	1.014	1.047	1.034
C_4/C_{20}	1.342	1.335	1.282	1.369	1.342	1.308	1.325

Table 3. Experimental rho length constant for various iterating functions.

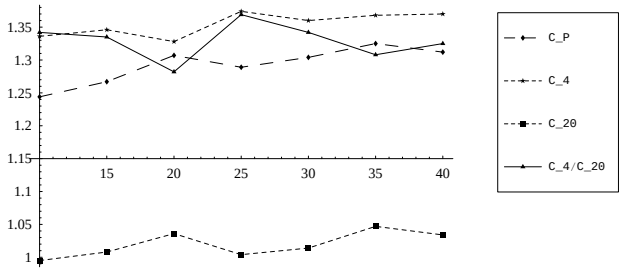


Fig. 1. Expected rho length constants for f_P, f_4, and f_{20}.

C_r and C_P, and this is summarized in Table 3. We have also provided graphs for some of these in Figure 1.

It is clear that our data is not very accurate, but it is good enough for one to conclude that C_4/C_{20} will not be too different from 1.3, even for large q.

B Tag Tracing on Binary Fields

Let us explain how tag tracing can be applied to cyclic subgroups of binary fields. We shall be very brief, as much of this case is quite similar to the prime field case.

Fix the binary field to $\mathbf{F}_{2^m} = \mathbf{F}_2[t]/p(t)$, where $p(t)$ is an irreducible polynomial of degree m, so that elements of the cyclic group $G \subset \mathbf{F}_{2^m}^{\times}$ may be written in the polynomial basis. Adopting the notation used with integers, we shall write

$\lfloor p_1(t)/p_2(t) \rfloor$ and $p_1(t) \bmod p_2(t)$ to denote the quotient and remainder, respectively, resulting from the polynomial division of $p_1(t)$ by $p_2(t)$.

We fix positive integers u and v, such that $v < u \leq \frac{m+1}{2}$, and define the polynomial $B(t) = \lfloor p(t)/t^u \rfloor$. The tag function $\tau : G \to \mathcal{T} = \{f \in \mathbf{F}_2[t] \mid \deg f < u - v\}$ is defined as

$$\tau\big(g(t)\big) = \left\lfloor \frac{g(t) \bmod p(t)}{t^v \cdot B(t)} \right\rfloor. \tag{6}$$

Note that this map is surjective and will be roughly pre-image uniform for usual choices of G.

Given an $\mathbf{x}(t) \in \mathbf{F}_2[t]$, we can write $\mathbf{x}(t) = \sum_i x_i(t) \cdot t^{(v+1)i}$, with $\deg x_i(t) \leq v$. Also, given $\mathbf{y}(t) \in \mathbf{F}_2[t]$, we can write, $t^{(v+1)i} \cdot \mathbf{y}(t) \bmod p(t) = \hat{y}_i(t) \cdot B(t) + \check{y}_i(t)$ with $\deg \check{y}_i(t) < m - u$, for each meaningful i. Using this notation, we define the auxiliary tag function as

$$\bar{\tau}\big(\mathbf{x}(t), \mathbf{y}(t)\big) = \left\lfloor \frac{\sum_i x_i(t) \cdot \hat{y}_i(t) \bmod t^u}{t^v} \right\rfloor. \tag{7}$$

Then, through careful counting of degrees and argument similar to the proof of Lemma 4, one can show that

$$\tau\big(\mathbf{x}(t) \cdot \mathbf{y}(t)\big) = \bar{\tau}\big(\mathbf{x}(t), \mathbf{y}(t)\big).$$

We emphasize that this is true for any choice of $\mathbf{x}(t), \mathbf{y}(t) \in \mathbf{F}_2[t]$.

Finally, we view the polynomial set $\mathcal{T}$ as the set of non-negative integers less than $|\mathcal{T}| = 2^{u-v}$ and define $\sigma : \mathcal{T} \to \mathcal{S} = \{0, \ldots, r-1\}$ to be division by $\lceil |\mathcal{T}|/r \rceil$. Then, the index function $s = \sigma \circ \tau : G \to \mathcal{S}$ is pre-image uniform for r that is a power of 2. For other r, the probability of reaching each of the indices may differ by at most $1/|\mathcal{T}|$.

In the binary field case, unlike the prime field case, the auxiliary tag function always gives the correct tag value, so one has a better chance of running through the full ℓ-many tag tracing steps with the pre-computed table $\mathcal{M}_\ell$, without fully computing any product. However, the asymptotic complexity of the binary field case remains equal to that of the prime field case.