MATHEMATICS OF COMPUTATION
Volume 80, Number 273, January 2011, Pages 477–500
S 0025-5718(10)02356-2
Article electronically published on April 16, 2010

STRUCTURE COMPUTATION AND DISCRETE LOGARITHMS IN FINITE ABELIAN p-GROUPS

ANDREW V. SUTHERLAND

Abstract. We present a generic algorithm for computing discrete logarithms in a finite abelian p-group H, improving the Pohlig–Hellman algorithm and its generalization to noncyclic groups by Teske. We then give a direct method to compute a basis for H without using a relation matrix. The problem of computing a basis for some or all of the Sylow p-subgroups of an arbitrary finite abelian group G is addressed, yielding a Monte Carlo algorithm to compute the structure of G using $O(|G|^{1/2})$ group operations. These results also improve generic algorithms for extracting pth roots in G.

1. Introduction

The discrete logarithm plays two opposing roles in group computations. As a constructive tool, discrete logarithms are the key ingredient in generic algorithms for extracting roots (including square roots in finite fields) [2, 23, 27, 31] and for computing group structure [7, 8, 26, 28, 30]. On the other hand, a wide range of cryptographic applications depend on the essential difficulty of computing discrete logarithms in the worst case (see [15] or [17] for a survey).

Typically, the discrete logarithm is defined in the context of a cyclic group: for any $\beta \in \langle\alpha\rangle$ there is a unique nonnegative integer $x < |\alpha|$ for which $\beta = \alpha^x$. More generally, given $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$, if every $\beta \in \langle\boldsymbol{\alpha}\rangle$ can be written uniquely as[1]

$$\beta = \boldsymbol{\alpha}^{\boldsymbol{x}} = \alpha_1^{x_1} \cdots \alpha_r^{x_r},$$

with $0 \le x_i < |\alpha_i|$, then $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \beta)$ is the *discrete logarithm* of β with respect to $\boldsymbol{\alpha}$, and we call the vector $\boldsymbol{\alpha}$ a *basis* for the group it generates. We work in the computational framework of generic group algorithms, as defined, for example, in [26]. Thus we suppose that a "black box" is used to perform group operations, possibly including the provision of random elements, with each group element arbitrarily assigned a unique identifier.

We are interested in constructive applications of the discrete logarithm, but let us first recall the negative result of Shoup [24]. Any generic algorithm to compute discrete logarithms in a finite abelian group G with prime exponent[2] uses $\Omega(|G|^{1/2})$ group operations. A matching upper bound is achieved, for cyclic groups, by Shanks' baby-step giant-step algorithm [22] and (probabilistically) by Pollard's

Received by the editor September 19, 2008 and, in revised form, July 27, 2009 and August 29, 2009.

2010 *Mathematics Subject Classification.* Primary 11Y16; Secondary 20K01, 12Y05.

[1]Consistent with our use of the word "logarithm", we write groups multiplicatively.

[2]The exponent of G is the least positive integer n for which $\alpha^n = 1_G$ for all $\alpha \in G$.

rho method [20, 29]. Both algorithms can be generalized to compute discrete logarithms in any finite abelian group using $O(|G|^{1/2})$ group operations [7, 26, 28].

However, when the exponent of the group is not prime, we can do better. This was proven for cyclic groups by Pohlig and Hellman [19] and later generalized by Teske [30].[3] The Pohlig–Hellman approach relies on computing discrete logarithms in subgroups of the given group. The reduction to subgroups of prime-power order is straightforward, hence we focus primarily on abelian p-groups.

If $\boldsymbol{\alpha}$ is a basis for a finite abelian group G of exponent p^m and rank r, Teske's generalization of the Pohlig–Hellman algorithm computes $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$ using

$$T_{\mathrm{DL}}(G) = O(m \lg |G| + m p^{r/2}) \tag{1}$$

group operations [30, Thm. 6.1].[4] When $m = 1$ this reduces to the $O(|G|^{1/2})$ upper bound mentioned above. If p and r are small (when computing square roots in finite fields, for example, $r = 1$ and $p = 2$) the first term dominates and the complexity becomes $O(n^2)$, where $n = \lg |G|$. For cyclic groups this can be improved to $O(n \lg n)$ [25, §11.2.3], and here we achieve an $O(n \lg n / \lg \lg n)$ bound for arbitrary finite abelian groups when p and r are suitably bounded. More generally, Algorithm 1 computes $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$ using

$$T_{\mathrm{DL}}(G) = O\left(\frac{\lg(m+1)}{\lg\lg(m+2)} \lg |G| + \frac{\log_p |G|}{r} p^{r/2}\right) \tag{2}$$

group operations, improving the dependence on m in both terms of (1).

Discrete logarithms may be applied to compute the structure of a finite abelian group. Typically, one uses discrete logarithms to construct a relation matrix, which is then reduced to yield a basis by computing the Smith normal form [7, 8, 28]. We take a simpler (and faster) approach, using our algorithm for discrete logarithms to directly construct a basis. Given a generating set S for a finite abelian p-group G of rank r, we give a deterministic generic algorithm to construct a basis using

$$T_{\mathrm{B}}(S) = O\left(\lg^{2+\epsilon} |G| + (|S| - r + 1) T_{\mathrm{DL}}(G)\right) \tag{3}$$

group operations, improving the $O(|S||G|^{1/2})$ result of Buchmann and Schmidt [8].

The bound in (3) is minimized when $|S| \approx r$. If we pick a random subset $S \subset G$, of size $r + O(1)$, then S generates G with very high probability [21]. When combined with an algorithm to compute the group exponent, this yields a generic Monte Carlo algorithm to compute the structure of an arbitrary finite abelian group using $O(|G|^{1/2})$ operations. When sufficiently tight bounds on the group order are known, this can be converted to a Las Vegas algorithm.

This approach can also be applied to a Sylow p-subgroup $H \subset G$. If the group exponent (or order) is known, the complexity then depends primarily on the size and shape of H, not G. This is useful when extracting pth roots in G, which only requires a basis for H [27].

2. Abelian p-groups and Young tableaux

We begin by describing a bijection between finite abelian p-groups and Young tableaux that motivates our approach and allows us to fix some terminology.

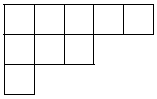
[3] Pohlig and Hellman credit Roland Silver, and also Richard Schroeppel and H. Block, for (unpublished) independent discovery of the same algorithm [19, p. 107].

[4] Teske actually addresses a more general problem, find the minimal nontrivial solution $(\boldsymbol{x}, y)$ to $\beta^y = \boldsymbol{\alpha}^{\boldsymbol{x}}$, which we consider in Section 4. Note that we use $\lg x = \log_2 x$ throughout.

We work in this section and the next with a basis $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$ for an abelian p-group G of order p^n, exponent p^m, and p-rank (rank) r. We let $|\alpha_i| = p^{n_i}$ and assume that $m = n_1 \geq \cdots \geq n_r \geq 1$. Up to isomorphism, G is determined by the integer partition $\pi(G) = (n_1, \ldots, n_r)$. For example, if

$$G \cong \mathbb{Z}/p^5\mathbb{Z} \times \mathbb{Z}/p^3\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}, \tag{4}$$

then $\pi(G) = (5, 3, 1)$ is a partition of $n = 9$ into three parts, with Young diagram:



A cyclic group has $r = 1$ and a single row in its diagram, while a group with prime exponent has $m = 1$ and a single column. In our example, G has $r = 3$ and $m = 5$.

For each $\beta \in G$, we regard $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \beta)$ as an element of the ring

$$R_{\boldsymbol{\alpha}} = R_{\alpha_1} \times \cdots \times R_{\alpha_r} = \mathbb{Z}/p^{n_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p^{n_r}\mathbb{Z}. \tag{5}$$

The additive group of $R_{\boldsymbol{\alpha}}$ is isomorphic to G, via the map $\boldsymbol{x} \rightsquigarrow \boldsymbol{\alpha}^{\boldsymbol{x}}$ (the inverse map sends β to $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$). We may write the components of $\boldsymbol{x} \in R_{\boldsymbol{\alpha}}$ in base p as

$$x_i = \sum_{j=1}^{n_i} p^{n_i - j} x_{i,j},$$

where $x_{i,1}$ is the *most* significant digit (and may be zero). We can then represent $\boldsymbol{x}$ (and $\beta = \boldsymbol{\alpha}^{\boldsymbol{x}}$) by a Young tableau of shape $\pi(G)$ with label $x_{i,j}$ in the ith row and jth column. For our example G in (4), if $p = 2$ and $\boldsymbol{x} = (13, 5, 1)$, we have

0	1	1	0	1
1	0	1		
1				

(6)

corresponding to $\beta = \boldsymbol{\alpha}^{\boldsymbol{x}} = \alpha_1^{13}\alpha_2^5\alpha_3$.

We wish to split the tableau above into left and right halves, allowing us to write

$$\boldsymbol{x} = \boldsymbol{q}\boldsymbol{v} + \boldsymbol{u}.$$

The vector $\boldsymbol{q}$ is a "shift" vector whose components are powers of p, while $\boldsymbol{v}$ and $\boldsymbol{u}$ correspond to the left and right halves of $\boldsymbol{x}$, respectively. These vectors are obtained by computing discrete logarithms in certain subgroups of G, as we now describe.

If we multiply $\boldsymbol{x}$ (exponentiate β) by the integer scalar p^k, this shifts the labels of the tableau to the left k places, leaving zeros on the right. In our example, if $k = 2$, we have $4\boldsymbol{x} = (20, 4, 0)$, yielding

1	**0**	**1**	0	0
1	0	0		
0				

(7)

(with shifted labels in bold), corresponding to $\beta^4 = \alpha_1^{20}\alpha_2^4$. The element β^{p^k} lies in the subgroup of p^kth powers in G,

$$G^{p^k} = \{\beta^{p^k} : \beta \in G\}, \tag{8}$$

which has a basis[5] $\boldsymbol{\gamma}$ defined by $\gamma_i = \alpha_i^{p^k}$. The diagram of G^{p^k} corresponds to the $m-k$ rightmost columns in the diagram of G. In our example we have the shape $\pi(G^4) = (3,1,0) = (3,1)$.

Now let $\boldsymbol{u} = \mathrm{DL}(\boldsymbol{\gamma}, \beta^{p^k})$. The vector $\boldsymbol{u}$ is an element of $R_{\boldsymbol{\gamma}}$, but as a vector of integers written in base p, each component of $\boldsymbol{u}$ contains the low order $m-k$ digits of the corresponding component of $\boldsymbol{x}$. We may "clear" these digits of $\boldsymbol{x}$ to obtain $\boldsymbol{z} \in R_{\boldsymbol{\alpha}}$ by subtracting $\boldsymbol{u}$ from $\boldsymbol{x}$ (in $\mathbb{Z}^r$), to obtain a reduced element of $R_{\boldsymbol{\alpha}}$. In our example we have $\boldsymbol{u} = (5,1,0)$, $\boldsymbol{z} = (8,4,1)$ and the tableau

0	**1**	0	0	0
1	**0**	0		
1				

(9)

with the entries unaffected by subtracting $\boldsymbol{u}$ from $\boldsymbol{x}$ in bold. The element $\beta\boldsymbol{\alpha}^{-\boldsymbol{u}}$ has order at most p^k and lies in the p^k-torsion subgroup

$$G[p^k] = \{\beta : \beta^{p^k} = 1_G,\ \beta \in G\}. \tag{10}$$

A basis $\boldsymbol{\delta}$ for $G[p^k]$ is given by $\delta_i = \alpha_i^{q_i}$, where $q_i = p^{\max(0,n_i-k)}$. The diagram of $G[p^k]$ corresponds to the k leftmost columns of the diagram of G. In our example, $\pi(G[4]) = (2,2,1)$. If we now let $\boldsymbol{v} = \mathrm{DL}(\boldsymbol{\delta}, \beta\boldsymbol{\alpha}^{-\boldsymbol{u}})$, then $\boldsymbol{z} = \boldsymbol{q}\boldsymbol{v}$ and

$$\boldsymbol{x} = \boldsymbol{q}\boldsymbol{v} + \boldsymbol{u},$$

as desired. In our example we have $\boldsymbol{q} = (8,2,1)$ and $\boldsymbol{v} = (1,2,1)$ yielding

$$(13,5,1) = (8,2,1)(1,2,1) + (5,1,0).$$

This equation effectively reconstructs the tableau in (6) by gluing together the bold portions of the tableaux in (7) and (9).

Note that $\boldsymbol{u}$ and $\boldsymbol{v}$ were defined via discrete logarithms in the subgroups G^{p^k} and $G[p^k]$, respectively. This suggests a recursive approach, leading to base cases in subgroups corresponding to single columns in the Young diagram of G.

3. Computing discrete logarithms

A recursive algorithm, along the lines suggested above, already yields an improvement over the result of Teske [28]; in the cyclic case this is equivalent to Shoup's balanced divide-and-conquer version of the Pohlig–Hellman algorithm [25, 11.2.3]. We can achieve a further speedup by broadening the recursion tree, allowing us to take advantage of fixed-base exponentiation techniques. At the same time, we may structure the algorithm to facilitate precomputation, an important practical optimization in applications that rely heavily on discrete logarithms [4, 27].

We will need to compute discrete logarithms in various subgroups of the form

$$G(j,k) = \{\beta^{p^j} : \beta^{p^k} = 1_G,\ \beta \in G\}, \tag{11}$$

for nonnegative integers $j < k$. The subgroup $G(j,k)$ consists of all p^jth powers of order at most p^{k-j} and corresponds to columns $j+1$ through k in the diagram of G. If G has exponent p^m, then $G = G(0,m)$.

We wish to obtain a basis for $G(j,k)$ from our given basis $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$ for G. To this end, let $n_i = \log_p |\alpha_i|$, let $q_i = p^{j+\max(0,n_i-k)}$, and define

$$\boldsymbol{q}(j,k) = (q_1, \ldots, q_r) \qquad \text{and} \qquad \boldsymbol{\alpha}(j,k) = \boldsymbol{\alpha}^{\boldsymbol{g}(j,k)} = (\alpha^{q_1}, \ldots, \alpha^{q_r}). \tag{12}$$

[5]Our definition of a basis allows $\boldsymbol{\gamma}$ to contain trivial elements. In practice we may truncate $\boldsymbol{\gamma}$.

Then $\boldsymbol{\alpha}(j,k)$ is our desired basis, as we now show.

Lemma 1. *Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$ be a basis for a finite abelian G, and let j and k be nonnegative integers with $j < k$. Then $\boldsymbol{\alpha}(j,k)$ is a basis for $G(j,k)$.*

Proof. Let $\boldsymbol{\gamma} = \boldsymbol{\alpha}(j,k)$. We first show that γ is a basis for $\langle\boldsymbol{\gamma}\rangle$. Suppose for the sake of contradiction that $\boldsymbol{\gamma}^{\boldsymbol{x}} = \boldsymbol{\gamma}^{\boldsymbol{y}}$ with $\boldsymbol{x}, \boldsymbol{y} \in R_{\boldsymbol{\gamma}}$ distinct. We must have $x_i \neq y_i$ for some i, which implies $\gamma_i \neq 1_G$ and $q_i|\gamma_i| = |\alpha_i|$. We also have $\boldsymbol{\alpha}^{\boldsymbol{q}(j,k)\boldsymbol{x}} = \boldsymbol{\alpha}^{\boldsymbol{q}(j,k)\boldsymbol{y}}$. As $q_i x_i$ and $q_i y_i$ are distinct integers less than $|\alpha_i|$, the vectors $\boldsymbol{q}(j,k)\boldsymbol{x}$ and $\boldsymbol{q}(j,q)\boldsymbol{y}$ are distinct elements of $R_{\boldsymbol{\alpha}}$. But this is a contradiction, since $\boldsymbol{\alpha}$ is a basis.

We now prove $\langle\boldsymbol{\gamma}\rangle = G(j,k)$. Every γ_i is a p^jth power (since $p^j|q_i$) and has order at most p^{k-j} (since $p^{n_i}|p^{k-j}q_i$); thus $\langle\boldsymbol{\gamma}\rangle \subset G(j,k)$. Conversely, for $\delta \in G(j,k)$, if $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \delta)$, then $p^j|x_i$ and $p^{n_i}|p^{k-j}x_i$ for each i. If $k < n_i$, then $p^{j+n_i-k}|x_i$, so $q_i|x_i$ in every case. It follows that $\delta \in \langle\boldsymbol{\gamma}\rangle$; hence $G(j,k) \subset \langle\boldsymbol{\gamma}\rangle$. □

We now give a recursive algorithm to compute discrete logarithms in $G(j,k)$, using $\boldsymbol{\alpha}(j,k)$ and $\boldsymbol{q}(j,k)$ as defined above. Note that if $j \leq j' < k' \leq k$, then each component of $\boldsymbol{q}(j,k)$ divides the corresponding component of $\boldsymbol{q}(j',k')$, and we may then write $\boldsymbol{q}(j',k')/\boldsymbol{q}(j,k)$ to denote point-wise division. For convenience, let $\mathrm{DL}_{\boldsymbol{\alpha}}(j,k,\beta)$ denote $\mathrm{DL}(\boldsymbol{\alpha}(j,k),\beta)$. We assume the availability of a standard algorithm for computing discrete logarithms in the base cases, as discussed below.

Algorithm 1. Given a basis $\boldsymbol{\alpha}$ for a finite abelian p-group G and $t \in \mathbb{Z}_{>0}$, the following algorithm computes $\mathrm{DL}_{\boldsymbol{\alpha}}(j,k,\beta)$ for integers $0 \leq j < k$ and $\beta \in G(j,k)$:

1. If $k - j \leq t$, compute $\boldsymbol{x} \leftarrow \mathrm{DL}_{\boldsymbol{\alpha}}(j,k,\beta)$ as a base case and return $\boldsymbol{x}$.
2. Choose integers $j_1, \ldots, j_w$ satisfying $j = j_1 < j_2 < \cdots < j_w < j_{w+1} = k$.
3. Compute $\gamma_i = \beta^{p^{j_i - j}}$ for i from 1 to w, and set $\boldsymbol{x} \leftarrow 0$.
4. For i from w down to 1:
   a. Recursively compute $\boldsymbol{v} \leftarrow \mathrm{DL}_{\boldsymbol{\alpha}}(j_i, j_{i+1}, \gamma_i \boldsymbol{\alpha}(j_i,k)^{-\boldsymbol{x}})$.
   b. Set $\boldsymbol{x} \leftarrow \boldsymbol{s}\boldsymbol{v} + \boldsymbol{x}$, where $\boldsymbol{s} = \boldsymbol{q}(j_i, j_{i+1})/\boldsymbol{q}(j_i,k)$.
5. Return $\boldsymbol{x}$.

Example 1. Let G be cyclic of order p^{19} with basis α, $j = 6$ and $k = 13$. Then $q(6,13) = p^{6+\max(0,19-13)} = p^{12}$ and $\alpha^{p^{12}}$ is a basis for $G(6,13)$. Let $j_2 = 8$ and $j_3 = 11$, so that $(6,13]$ is partitioned into subintervals $(6,8]$, $(8,11]$, and $(11,13]$. We then have $q(11,13) = p^{17}$, $q(8,11) = p^{16}$, $q(6,8) = p^{17}$, and also $q(8,13) = p^{14}$. For $\beta \in G(6,13)$, Algorithm 1 computes

$$\begin{aligned} v_3 &= \mathrm{DL}(\alpha^{p^{17}}, \beta^{p^5}), & x_3 &= v_3, \\ v_2 &= \mathrm{DL}(\alpha^{p^{16}}, \beta^{p^2}\alpha^{-p^{14}x_3}), & x_2 &= p^2 v_2 + v_3, \\ v_1 &= \mathrm{DL}(\alpha^{p^{17}}, \beta\alpha^{-p^{12}x_2}), & x_1 &= p^5 v_1 + p^2 v_2 + v_3. \end{aligned}$$

The final value $x = x_1$ contains 7 base p digits: 2 in v_1, 3 in v_2, and 2 in v_3.

Example 2. Suppose instead that G is cyclic of order p^9, but keep the other parameters as above. We then have $q(6,13) = p^6$, $q(11,13) = p^{11}$, $q(8,11) = p^8$, $q(6,8) = p^7$, and $q(8,13) = p^8$. For $\beta \in G(6,13)$, the algorithm now computes

$$\begin{aligned} v_3 &= \mathrm{DL}(1_G, 1_G), & x_3 &= v_3 = 0, \\ v_2 &= \mathrm{DL}(\alpha^{p^8}, \beta^{p^2}\alpha^{-p^8 x_3}), & x_2 &= v_2, \\ v_1 &= \mathrm{DL}(\alpha^{p^7}, \beta\alpha^{-p^6 x_2}), & x_1 &= p v_1 + v_2. \end{aligned}$$

The computation of x_3 requires no group operations; the algorithm can determine $\alpha(11,13) = 1_G$ from the fact that $11 \geq 9$ (since $|\alpha| = p^9$ is given). The final value $x = x_1$ contains 3 base p digits: 2 in v_1 and 1 in v_2.

These examples illustrate the general situation; we compute discrete logarithms in r cyclic groups in parallel. The second example is contrived, but it shows what happens when a cyclic factor of G has order less than p^k.

We assume that no cost is incurred by trivial operations (those involving the identity element). As a practical optimization, the loop in step 4 may begin with the largest i for which $\gamma_i \neq 1_G$ (it will compute $\boldsymbol{x} = 0$ up to this point in any event).

The correctness of Algorithm 1 follows inductively from the lemma below.

Lemma 2. *Let $\boldsymbol{\alpha}$ be a basis for a finite abelian p-group G and let j, j', k', and k be integers with $0 \le j \le j' < k' \le k$. For all $\beta \in G(j,k)$ the following hold:*

(i) *If $\boldsymbol{x} = \mathrm{DL}_{\boldsymbol{\alpha}}(k', k, \beta^{p^{k'-j}})$ and $\gamma = \beta^{p^{j'-j}}\boldsymbol{\alpha}(j',k)^{-\boldsymbol{x}}$, then $\gamma \in G(j',k')$.*

(ii) *If we also have $\boldsymbol{v} = \mathrm{DL}_{\boldsymbol{\alpha}}(j', k', \gamma)$ and $\boldsymbol{s} = \boldsymbol{q}(j',k')/\boldsymbol{q}(j',k)$, then $\boldsymbol{s}\boldsymbol{v} + \boldsymbol{x} = \mathrm{DL}_{\boldsymbol{\alpha}}(j', k, \beta^{p^{j'-j}})$.*

Proof. For (i), note that β is a p^jth power, so $\beta^{p^{j'-j}}$ is a $p^{j'}$th power, and every element of $\langle \boldsymbol{\alpha}(j',k)\rangle$ is a $p^{j'}$th power, hence γ is a $p^{j'}$th power. We also have

$$\gamma^{p^{k'-j'}} = \left(\beta^{p^{j'-j}}\boldsymbol{\alpha}(j',k)^{-\boldsymbol{x}}\right)^{p^{k'-j'}} = \beta^{p^{k'-j}}\boldsymbol{\alpha}(j',k)^{-p^{k'-j'}\boldsymbol{x}}.$$

It follows from the definition in (12) that $\boldsymbol{\alpha}(j',k)^{-p^{k'-j'}\boldsymbol{x}} = \boldsymbol{\alpha}(k',k)^{-\boldsymbol{x}}$, since we have $j' + \max(0, n_i - k) + k' - j' = k' + \max(0, n_i - k)$. We then obtain

$$\gamma^{p^{k'-j'}} = \beta^{p^{k'-j}}\boldsymbol{\alpha}(k',k)^{-\boldsymbol{x}} = \beta^{p^{k'-j}}(\beta^{p^{k'-j}})^{-1} = 1_G.$$

Thus γ has order at most $p^{k'-j'}$, and therefore $\gamma \in G(j',k')$, proving (i).

Note that $k' < k$ implies $\max(0, n_i - k') \ge \max(0, n_i - k)$, so $\boldsymbol{q}(j',k')$ is divisible (component-wise) by $\boldsymbol{q}(j',k)$, and $\boldsymbol{s}$ in (ii) is well defined. Now

$$\boldsymbol{\alpha}(j',k)^{\boldsymbol{s}\boldsymbol{v}} = \boldsymbol{\alpha}^{\boldsymbol{q}(j',k)\boldsymbol{s}\boldsymbol{v}} = \boldsymbol{\alpha}^{\boldsymbol{q}(j',k')\boldsymbol{v}} = \boldsymbol{\alpha}(j',k')^{\boldsymbol{v}} = \gamma = \beta^{p^{j'-j}}\boldsymbol{\alpha}(j',k)^{-\boldsymbol{x}},$$

and therefore $\boldsymbol{\alpha}(j',k)^{\boldsymbol{s}\boldsymbol{v}+\boldsymbol{x}} = \beta^{p^{j'-j}}$, proving (ii). □

We now consider the parameter t in Algorithm 1. If p^r is small, we precompute a lookup table for $G(0,t)$, containing at most p^{rt} group elements, for some suitable value of t. This will handle all the base cases, since they arise in subgroups $G(j,k)$ of $G(0,t)$, where $k - j \le t$. This is especially effective when one can amortize the cost over many discrete logarithm computations, in which case a larger t is beneficial. In applications where $p^r = O(1)$, one typically chooses t to be logarithmic in the relevant problem size (which may be larger than $|G|$).

When p^r is large, we instead set $t = 1$ and use a standard $O(\sqrt{N})$ algorithm for computing discrete logarithms in finite abelian groups. A space-efficient algorithm derived from Pollard's rho method is given in [28], and a baby-steps giant-steps variant can be found in [26, Alg. 9.3] (see Section 6 for optimizations).

When partitioning the interval $(j,k]$ into subintervals in step 2, we assume that the subintervals are of approximately equal size, as determined by the choice of w. The choice $w = k - j$ limits the recursion depth to 1 and corresponds to the standard Pohlig–Hellman algorithm. The choice $w = 2$ yields a balanced binary recursion tree. This might appear to be an optimal choice, but we can actually do better with a somewhat larger choice of w, using fixed-base exponentiation techniques.

We recall a theorem of Yao.

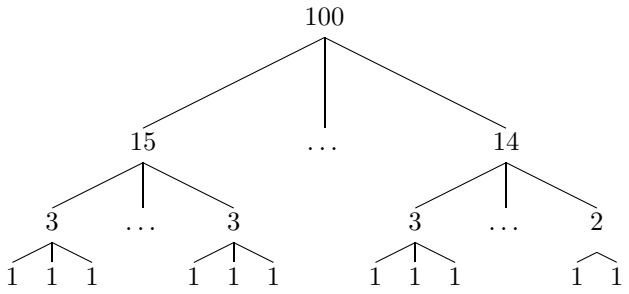
Theorem 1 (Yao[6]). *There is an online algorithm that, given $\gamma \in G$ and any input sequence of positive integers $e_1, \ldots, e_w$, outputs $\gamma^{e_1}, \ldots, \gamma^{e_w}$ using at most*

$$\lg E + c \sum_{i=1}^{w} \left\lceil \frac{\lg e_i}{\lg \lg (e_i + 2)} \right\rceil$$

multiplications, where $E = \max_i\{e_i\}$ and $c \le 2$ is a constant.[7]

The online algorithm in the theorem outputs γ^{e_i} before receiving the input e_{i+1}. If we set $n = \lg E$, Yao's Theorem tells us that, provided $w = O(\lg n)$, we can perform w exponentiations of a common base with n-bit exponents using just $O(n)$ multiplications, the same bound as when $w = 1$. There are several algorithms that achieve Yao's bound [5, 6, 12, 14, 16], and they typically require storage for $O(n/\lg n)$ group elements.

Consider the execution of Algorithm 1 computing $\mathrm{DL}(\boldsymbol{\alpha}, \beta) = \mathrm{DL}(0, m, \beta)$. It will be convenient to label the levels of the recursion tree with $\ell = 0$ at the bottom and $\ell = d$ at the top, where d is the maximum depth of the recursion. At each level the interval $(0, m]$ is partitioned into successively smaller subintervals. We let $s_d = m$ denote the size of the initial interval at level d, and at level ℓ we partition each interval into approximately w_ℓ subintervals of maximum size $s_{\ell-1} = \lceil s_\ell / w_\ell \rceil$ and minimum size $\lfloor s_\ell / w_\ell \rfloor$.



In the tree above we start at level $\ell = 3$ with $s_3 = m = 100$ and $w_3 = 7$, partitioning 100 into two subintervals of size $s_2 = 15$ and five subintervals of size 14. We then have $w_2 = 5$ and $s_1 = 3$, and finally $w_1 = 3$ and $s_0 = 1$. The base cases are all at level 0 in this example, but in general may also occur at level 1. The fan-out of each node at level ℓ is w_ℓ, except possibly at level 1 (in this example, we cannot partition 2 into three parts).

Our strategy is to choose $w_\ell \approx \min(\lg(s_\ell \lg p), s_\ell)$ and apply Yao's Theorem to bound the cost at each level of the recursion tree by $O(\log |G|)$ group operations, not including the base cases. The standard Pohlig–Hellman approach reduces the problem to base cases in one level, potentially incurring a cost of $O(\log^2 |G|)$ to do so. A binary recursion uses $O(\lg |G|)$ group operations at each level, but requires $\Omega(\lg m)$ levels, while we only need $O(\lg m / \lg \lg m)$.

With these ideas in mind, we now prove an absolute bound on the running time of Algorithm 1. An asymptotic bound appears in the corollary that follows.

[6] Pippenger gives a better bound for large w, but not necessarily an online algorithm [5, 18].
[7] As $E \to \infty$ the constant c can be made arbitrarily close to 1.

Proposition 1. *Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$ be a basis for a finite abelian p-group G with rank r and exponent p^m. Set $n_i = \log_p |\alpha_i|$, and let r_j be the rank of the subgroup of p^jth powers in G. There is a generic algorithm to compute $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$ using*

$$T_{\mathrm{DL}}(G) \le c \left(\sum_{i=1}^{r} \frac{\lg(n_i+1)}{\lg\lg(n_i+2)} \lg|\alpha_i| + \sum_{j=0}^{m-1} p^{r_j/2} \right)$$

group operations, where c is an absolute constant independent of G.

When a probabilistic algorithm is used for the base cases (such as the rho method), the algorithm in the proposition is probabilistic and $T_{\mathrm{DL}}(G)$ refers to the expected running time, but otherwise the algorithm is deterministic. The bound on $T_{\mathrm{DL}}(G)$ depends only on the structure of G, not the basis $\boldsymbol{\alpha}$.

Proof of Proposition 1. We use Algorithm 1 to compute $\mathrm{DL}(\boldsymbol{\alpha}, \beta) = \mathrm{DL}_{\boldsymbol{\alpha}}(0, m, \beta)$ using $t = 1$. As discussed above, we label the levels of the recursion tree with $\ell = 0$ at the base and $\ell = d$ at the root. We let s_ℓ denote the size of the first interval at level ℓ, and we assume that all others have size at least $s_\ell - 1$ and at most s_ℓ. Thus $s_d = m$, and we recursively define $s_{\ell-1} = \lceil s_\ell / w_\ell \rceil$ down to $s_0 = 1$. To simplify the proof we use $w_\ell = \lceil \lg(2s_\ell) \rceil$ (independent of p) and assume $n_1 = m$.

There is a base case in $G(j, j+1)$ for each $0 \le j < m$, with $|G(j, j+1)| = p^{r_j}$. Applying either of the standard $O(\sqrt{N})$ discrete logarithm algorithms to the base cases yields the second sum in the bound for $T_{\mathrm{DL}}(G)$, with $c \approx 2$.

At level ℓ of the recursion tree, the total cost of step 3 is bounded by

$$T_1(\ell) < m(2 \lg p) = 2 \lg|\alpha_1|,$$

since m exponentiations by p are required. The total cost of the multiplications by γ_i in step 4a is bounded by $T_2(\ell) = m \le \lg|\alpha_1|$.

All other group operations occur in exponentiations in step 4a. These are of the form $\boldsymbol{\alpha}(j', k)^{-\boldsymbol{x}}$, where $j \le j' < k$. To bound their cost, we consider the cost $T(\alpha_i, \ell)$ associated to a particular α_i at level ℓ. The exponentiation of α_i is nontrivial only when $j' < n_i$ and $x_i \ne 0$. Thus to bound $T(\alpha_i, \ell)$, we only count exponentiations with $j' < n_i$ and only consider levels ℓ of the recursion tree with $s_{\ell-1} < n_i$, since at all higher levels x_i will still be zero.

In the recursive call to compute $\mathrm{DL}_{\boldsymbol{\alpha}}(j, k, \beta)$, we may compute $\boldsymbol{\alpha}(j', k)^{-\boldsymbol{x}}$ using fixed bases $\alpha_i^{-q_i}$, where $\boldsymbol{q} = \boldsymbol{q}(j, k)$ as in (12), since $\boldsymbol{q}(j, k)$ divides $\boldsymbol{q}(j', k)$ for all $j' \ge j$. For each α_i we can precompute all the $\alpha_i^{-q_i}$ for a cost of

$$T_0(\alpha_i) < 2 n_i \lg p = 2 \lg|\alpha_i|.$$

At level $\ell > 0$ with $s_{\ell-1} < n_i$, there are $\lceil n_i / s_l \rceil$ instances of up to $w_\ell - 1$ nontrivial exponentiations involving α_i. These are computed using the common base $\alpha_i^{-q_i}$, with exponents bounded by $E = \min(p^{s_\ell}, |\alpha_i|)$. Applying Yao's Theorem,

$$T(\alpha_i, \ell) \le \lceil n_i / s_l \rceil \left(\lg E + 2(w_\ell - 1) \left\lceil \frac{\lg E}{\lg\lg(E+2)} \right\rceil \right).$$

If $s_\ell > n_i$, we replace $\lceil n_i / s_\ell \rceil$ by 1 and $\lg E$ by $\lg|\alpha_i|$; otherwise we replace $\lg E$ by $s_\ell \lg p$. We then apply $\lceil z \rceil \le 2z$ (for $z \ge 1$) to remove both ceilings and obtain

$$T(\alpha_i, \ell) \le 2 \lg|\alpha_i| + 8 \lg|\alpha_i| \left(\frac{w_\ell - 1}{\lg\lg(E+2)} \right). \tag{13}$$

If $E = p^{s_\ell}$, then $(w_\ell - 1)/\lg\lg(E+2) < 2$. Otherwise $E = |\alpha_i|$, and then

$$\lg(E+2) > n_i \geq s_{\ell-1} + 1 = \lceil s_\ell / w_\ell \rceil + 1 \geq s_\ell/(\lg s_\ell + 2) + 1,$$

which implies $(w_\ell - 1)/\lg\lg(E+2) < 3$. This yields $T(\alpha_i, \ell) < 26 \lg|\alpha_i|$.

It remains to bound the number of levels $\ell > 0$ with $s_{\ell-1} < n_i$. This is equal to the least ℓ for which $s_\ell \geq n_i$, which we denote $d(n_i)$. We derive an upper bound on $d(n_i)$ as a function of n_i by proving a lower bound on s_ℓ as a function of ℓ.

We recall that $s_0 = 1$, and for $\ell > 0$ we have $w_\ell = \lceil \lg(2s_\ell) \rceil \geq 2$ and $s_\ell \geq 2s_{\ell-1}$. This implies $w_\ell \geq l + 1$ and $s_\ell \geq \ell s_{\ell-1} \geq \ell!$ for all $\ell > 0$. Stirling's formula yields a lower bound on s_ℓ, from which one obtains the upper bound

$$d(n_i) \leq \frac{2\lg(n_i+1)}{\lg\lg(n_i+2)}, \tag{14}$$

valid for $n_i \geq 14$. The lexicographically minimal sequence of integers satisfying $s_0 = 1$ and $s_{\ell-1} = \lceil s_\ell / \lceil \lg(2s_\ell) \rceil \rceil$ for all $\ell > 0$ begins at $1, 2, 4, 16, 121, 1441, \ldots$, and one finds that $s_1, \ldots, s_{13}$ satisfy (14), with $\ell = n_i$ and $s_\ell = d(n_i)$.

The total cost of all computations outside of the base cases is then bounded by

$$\sum_{\ell=1}^{d} \Big(T_1(\ell) + T_2(\ell) \Big) + \sum_{i=1}^{r} \left(T_0(\alpha_i) + \sum_{\ell=1}^{d(n_i)} T(\alpha_i, \ell) \right) \;\leq\; c \sum_{i=1}^{r} \frac{\lg(n_i+1)}{\lg\lg(n_i+2)} \lg|\alpha_i|,$$

where we use $d = d(n_1)$, and the constant $c < 57$. This yields the first sum in the bound for $T_{\text{DL}}(G)$ and completes the proof. □

For the sake of brevity we have overestimated the constant c in the proof above. Empirically, c is always less than 2 and is typically close to 1 (see Section 6).

The space used by Algorithm 1 depends on how the base cases are handled, but it can be bounded by $O(\lg|G| / \lg\lg|G|)$ group elements. There are at most $\sum_{i=1}^{r} d(n_i)\sqrt{n_i}$ distinct $\alpha_i^{-q_i}$ that need to be precomputed, which fits within this bound. In practice, additional precomputation using slightly more storage, perhaps $O(\lg|G|)$ elements, can accelerate both the exponentiations and the base cases [27].

For many groups arising "in nature", both sums in the bound for $T_{\text{DL}}(G)$ are typically dominated by their first terms. Divisor class groups of curves and ideal class groups of number fields are, at least heuristically, two examples. One often sees an L-shaped Young diagram, with $n \approx m + r$, where $n = \log_p|G|$. Algorithm 1 yields a useful improvement here, with a complexity of $O(p^{r/2})$ versus $O(mp^{r/2})$. More generally, we have the following corollary.

Corollary 1. *Let $\boldsymbol{\alpha}$ be a basis for a finite abelian p-group G of size p^n, exponent p^m, and rank r. There is a generic algorithm to compute* $\text{DL}(\boldsymbol{\alpha}, \beta)$ *using*

$$T_{\text{DL}}(G) = O\left(\frac{\lg(m+1)}{\lg\lg(m+2)} \lg|G| + \frac{n}{r} p^{r/2} \right)$$

group operations.

Proof. The first term is immediate from Proposition 1 since $\lg G = \sum_{i=1}^{r} \lg|\alpha_i|$ and $n_i \leq m$ for all i. For the second term, consider $\sum_{i=0}^{m-1} p^{r_i/2}$. We have $r = r_0 \geq r_i$ and $n = r_0 + \cdots + r_{m-1}$. For fixed r and n, the worst case, up to a constant factor, occurs when the r_i are roughly equal (one uses Lemma 9 to prove this). □

The asymptotic upper bound on $T_{\text{DL}}(G)$ achieved here is nearly tight for generic algorithms. When $r = O(n)$, the bound in the corollary becomes $O(p^{r/2})$, matching Shoup's $\Omega(p^{r/2})$ lower bound. Even when this is not the case, one may argue, along the lines of Shoup, that the sum $\sum p^{r_i/2}$ in Proposition 1 is tight in any event. The first term in the corollary is $O(\lg^{1+\epsilon}|G|)$, and one does not expect to do better than $O(\lg|G|)$, since we have an $\Omega(\lg|G|)$ lower bound for exponentiation.

To complete our discussion of discrete logarithms, we give an algorithm to compute $\text{DL}(\boldsymbol{\alpha}, \beta)$ in an arbitrary finite abelian group G. We assume that $\boldsymbol{\alpha}$ is a prime-power basis for G, composed of bases $\boldsymbol{\alpha}_p$ for each of the Sylow p-subgroups of G. The construction of such a basis is discussed in the next section (and readily obtained from a given basis in any event).

Algorithm 2. Given a prime-power basis $\boldsymbol{\alpha}$ for a finite abelian group G with $|G| = N = q_1 \cdots q_k$ a factorization into powers of distinct primes $p_1, \ldots, p_k$, and $\beta \in G$, the following algorithm computes $\boldsymbol{x} = \text{DL}(\boldsymbol{\alpha}, \beta)$:

1. Let $M_j = N/q_j$ and compute $\beta_j \leftarrow \beta^{M_j}$ for j from 1 to k.
2. Compute $\boldsymbol{x}_j \leftarrow \text{DL}(\boldsymbol{\alpha}_{p_j}, \beta_j)$ using Algorithm 1.
3. Set $\boldsymbol{x} \leftarrow \boldsymbol{x}_1/M_1 \circ \cdots \circ \boldsymbol{x}_k/M_k$.

The symbol "$\circ$" denotes concatenation of vectors. Since $\beta_j^{q_j} = 1_G$, we must have $\beta_j \in \langle \boldsymbol{\alpha}_{p_j} \rangle$, and the components of $\boldsymbol{\alpha}_{p_j}$ all have order a power of p_j. The exponent vector $\boldsymbol{x}_j$ is thus divisible by M_j, since M_j is coprime to p_j and therefore a unit in each factor of the ring $R_{\boldsymbol{\alpha}_{p_j}}$. The correctness of Algorithm 2 follows easily.

Let $n = \lg N$. For $k = O(\lg n)$, the exponentiations in step 1 can be performed using $O(n)$ group operations, by Yao's Theorem. As k approaches n, this bound increases to $O(n^2/(\lg n)^2)$, and one should instead apply the $O(n \lg n/\lg\lg n)$ algorithm of [26, Alg. 7.4]. The total running time is then

$$T_{\text{DL}}(G) = O\left(\frac{\lg(k+1)}{\lg\lg(k+2)} \lg|G|\right) + \sum_{j=1}^{k} T_{\text{DL}}(G_{p_j}) \tag{15}$$

group operations, where G_{p_j} denotes the Sylow p_j-subgroup of G. For sufficiently large $|G|$, the bound for $T_{\text{DL}}(G)$ is dominated by the sum in (15).

4. Constructing a basis for a finite abelian p-group

For a finite abelian group G, the *group structure* problem asks for a factor decomposition of G into cyclic groups of prime-power order, with a generator for each factor. This is equivalent to computing a basis for each of the (nontrivial) Sylow p-subgroups of G. We first suppose that G is a p-group and then give a reduction for the general case in Section 5.

Typically, a basis is derived from a matrix of relations among elements of a generating set for the group [7, 8, 9, 28]. This generating set may be given, or obtained (with high probability) from a random sample. One then computes the Smith normal form of the relation matrix [11, §2.4], applying corresponding group operations to the generating set to produce a basis.

Relations may be obtained via *extended discrete logarithms*. If $\boldsymbol{\alpha}$ is a basis for a subgroup of G and $\beta \in G$, then $\text{EDL}(\boldsymbol{\alpha}, \beta)$ is the pair $(\boldsymbol{x}, y)$ satisfying $\beta^y = \boldsymbol{\alpha}^{\boldsymbol{x}}$ that minimizes $y > 0$, with $\boldsymbol{x} \in R_{\boldsymbol{\alpha}}$. In a p-group, y is necessarily a power of p.

While our approach does not require us to compute $\mathrm{EDL}(\boldsymbol{\alpha}, \beta)$, we note that any algorithm for $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$ can be used to compute $\mathrm{EDL}(\boldsymbol{\alpha}, \beta)$.

Lemma 3. *Given a basis $\boldsymbol{\alpha}$ for a subgroup of a finite abelian p-group G and $\beta \in G$, there is a generic algorithm to compute $\mathrm{EDL}(\boldsymbol{\alpha}, \beta)$ using at most*

$$\left\lceil \lg(\log_p |\beta|) \right\rceil T_{\mathrm{DL}}(G) + 2 \lg |\beta|$$

group operations.

Proof. Assume Algorithm 1 returns an error whenever a base case fails.[8] Compute β^{p^j} for $0 \le j \le \log_p |\beta|$. Then use a binary search to find the least j for which one can successfully compute $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \beta^{p^j})$. We then have $\mathrm{EDL}(\boldsymbol{\alpha}, \beta) = (\boldsymbol{x}, p^j)$. □

Teske gives an algorithm to directly compute $\mathrm{EDL}(\boldsymbol{\alpha}, \beta)$, avoiding the $\lg(\log_p |\beta|)$ factor above, but this may still be slower than applying Lemma 3 to Algorithm 1. Alternatively, we may modify Algorithm 1 to solve a slightly easier problem. Instead of solving $\beta^y = \boldsymbol{\alpha}^{\boldsymbol{x}}$, we seek a solution to $\beta^y = \boldsymbol{\alpha}^{y\boldsymbol{x}}$.

More specifically, let us define a function $\mathrm{DL}^*_{\boldsymbol{\alpha}}(j, k, \beta)$ that extends the function $\mathrm{DL}_{\boldsymbol{\alpha}}(j, k, \beta)$ computed by Algorithm 1. If $\boldsymbol{\alpha}$ is a basis for a subgroup of an abelian p-group G and $\beta \in G$, we wish to compute a pair $(\boldsymbol{x}, h)$, with $\boldsymbol{x} = \mathrm{DL}_{\boldsymbol{\alpha}}(j+h, k, \beta^{p^h})$ and $h \ge 0$ minimal. It may be that there is no $h < k - j$ for which such a pair exists, and in this case[9] we let $h = k - j$ and $\boldsymbol{x} = 0$. When $\boldsymbol{\alpha}$ generates a subgroup with exponent p^m, we use $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$ to denote $\mathrm{DL}^*_{\boldsymbol{\alpha}}(0, m, \beta)$.

Algorithm 3. Let $\boldsymbol{\alpha}$ be a basis for a subgroup of a finite abelian p-group G and let $t \in \mathbb{Z}_{>0}$. Given $\beta \in G$ and $0 \le j < k$, compute $(\boldsymbol{x}, h) = \mathrm{DL}^*_{\boldsymbol{\alpha}}(j, k, \beta)$ as follows:

1. If $k - j \le t$, compute $(\boldsymbol{x}, h) \leftarrow \mathrm{DL}^*_{\boldsymbol{\alpha}}(j, k, \beta)$ as a base case. Return $(\boldsymbol{x}, h)$.
2. Choose integers $j_1, \ldots, j_w$ satisfying $j = j_1 < j_2 < \cdots < j_w < j_{w+1} = k$.
3. Compute $\gamma_i = \beta^{p^{j_i - j}}$ for i from 1 to w, and set $\boldsymbol{x} \leftarrow 0$.
4. For i from w down to 1:
   a. Recursively compute $(\boldsymbol{v}, h) \leftarrow \mathrm{DL}^*_{\boldsymbol{\alpha}}(j_i, j_{i+1}, \gamma_i \boldsymbol{\alpha}(j_i, k)^{-\boldsymbol{x}})$.
   b. Set $\boldsymbol{x} \leftarrow \boldsymbol{s}\boldsymbol{v} + \boldsymbol{x}$, where $\boldsymbol{s} = \boldsymbol{q}(j_i + h, j_{i+1})/\boldsymbol{q}(j_i + h, k)$.
   c. If $h > 0$, then return $(\boldsymbol{x}, j_i + h)$.
5. Return $(\boldsymbol{x}, 0)$.

For $t = 1$, the base case simply computes $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \beta)$ and returns $(\boldsymbol{x}, 0)$, or $(\boldsymbol{0}, 1)$ if a failure occurs. When $t > 1$, one applies Lemma 3 (if a lookup table is used, this means $O(\lg t)$ table lookups and $O(t \lg p)$ group operations).

Aside from the computation of h and the possibility of early termination, Algorithm 3 is essentially the same as Algorithm 1. Indeed, assuming $t = 1$, if $(\boldsymbol{x}, h)$ is the output of Algorithm 3, the sequence of group operations performed by Algorithm 1 on input $\boldsymbol{\alpha}^{\boldsymbol{x}}$ will be effectively identical (ignoring operations involving the identity). Thus the complexity bounds in Proposition 1 and its corollary apply.

To verify the correctness of Algorithm 3, we first note that if $h = k - j$, then the first base case must have failed and the output $(\boldsymbol{0}, h)$ is correct. If $h < k - j$, then it follows from the correctness of Algorithm 1 that $\boldsymbol{x} = \mathrm{DL}_{\boldsymbol{\alpha}}(j + h, k, \beta^{p^h})$. It is only necessary to check that h is minimal, but if not, the base case $\mathrm{DL}_{\boldsymbol{\alpha}}(j+h-1, j+h, \beta')$ would have succeeded and h would be smaller.

[8] Failure detection with baby-steps giant-steps or lookup table is easy. See [28] for a rho search.

[9] Arguably, h should be $\log_p |\beta|$ here (so that $\beta^{p^h} = \boldsymbol{\alpha}^{p^h \boldsymbol{x}}$), but this is less convenient.

We now explain how to construct a basis using Algorithm 3. Let us start with a vector $\boldsymbol{\alpha}$ consisting of a single element of G. Clearly $\boldsymbol{\alpha}$ is a basis for the cyclic subgroup it generates, and we would like to extend $\boldsymbol{\alpha}$ to a basis for all of G by adding elements to it one by one. This will only be possible if our basis at each step generates a subgroup H that is a factor of G (meaning $G \cong H \times G/H$). Some care is required, since H need not be a factor of G, but let us first consider how to extend a basis.

Given a basis $\boldsymbol{\alpha}$ for a subgroup H of G, we say that $\gamma \in G$ is *independent* of $\boldsymbol{\alpha}$ if the vector $\boldsymbol{\alpha} \circ \gamma = (\alpha_1, \dots, \alpha_r, \gamma)$ is a basis for $\langle \boldsymbol{\alpha}, \gamma \rangle$, and we write $\gamma \perp \boldsymbol{\alpha}$. The following lemma shows how and when one may use $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$ to obtain such a γ.

Lemma 4. *Let $\boldsymbol{\alpha}$ be a basis for a subgroup of a finite abelian p-group G, with $n_i = \log_p |\alpha_i|$, $m_0 = \min n_i$, and $m = \max n_i$. Let $\beta \in G$ and let $\gamma = \beta \boldsymbol{\alpha}^{-\boldsymbol{x}}$, where $(\boldsymbol{x}, h) = \mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$. The following hold:*

(i) *If $|\beta| \le p^m$, then $|\gamma| = p^h$.*
(ii) *If $|\beta| \le p^m$ and $|\gamma| \le p^{m_0}$, then $\gamma \perp \boldsymbol{\alpha}$.*

Proof. If $h < m$, then $\boldsymbol{x} = \mathrm{DL}^*_{\boldsymbol{\alpha}}(h, m, \beta)$, and we have

$$\beta^{p^h} = \boldsymbol{\alpha}(h,m)^{\boldsymbol{x}} = \boldsymbol{\alpha}^{\boldsymbol{q}(h,m)\boldsymbol{x}} = \boldsymbol{\alpha}^{p^h \boldsymbol{x}},$$

since $\boldsymbol{q}(h,m)$, as defined in (12), has $q_i = p^{\max(h, n_i - m - h)} = p^h$. It follows that

$$\gamma^h = (\beta \boldsymbol{\alpha}^{-\boldsymbol{x}})^{p^h} = 1_G,$$

and this cannot hold for any $h' < h$, by the minimality of h. Thus (i) holds when $h < m$. Now suppose $h = m$. Then $\boldsymbol{x} = \boldsymbol{0}$, $\gamma = \beta$, and $|\gamma| = |\beta| \ge p^h = p^m$. If $|\beta| \le p^m$, then $|\gamma| = |\beta| = p^h$; thus (i) also holds when $h = m$.

To prove (ii), assume $|\beta| \le p^m|$ and $|\gamma| \le p^{m_0}$, and suppose $\gamma \perp \boldsymbol{\alpha}$ does not hold. Then there is a nontrivial relation of the form $\gamma^{p^j} = \boldsymbol{\alpha}^{\boldsymbol{z}}$, for some $\boldsymbol{z} \in R_{\boldsymbol{\alpha}}$ and $j < h$, since $|\gamma| = p^h$ by (i). We claim that $\boldsymbol{z}$ is not divisible by p^j, since

$$\gamma^{p^j} = (\beta \boldsymbol{\alpha}^{-\boldsymbol{x}})^{p^j} = \beta^{p^j} \boldsymbol{\alpha}^{-p^j \boldsymbol{x}} = \boldsymbol{\alpha}^{\boldsymbol{z}},$$

and if p^j divides $\boldsymbol{z}$ we can set $\boldsymbol{v} = \boldsymbol{x} + \boldsymbol{z}/p^j$ to obtain

$$\beta^{p^j} = \boldsymbol{\alpha}^{\boldsymbol{z} + p^j \boldsymbol{x}} = (\boldsymbol{\alpha}^{\boldsymbol{v}})^{p^j} = \boldsymbol{\alpha}^{\boldsymbol{q}(j,m)\boldsymbol{v}} = \boldsymbol{\alpha}(j,m)^{\boldsymbol{v}},$$

which contradicts the minimality of h. We now note that if $|\gamma| \le p^{m_0}$, then γ^{p^j} has order at most $p^{m_0 - j}$. But $\boldsymbol{\alpha}^{\boldsymbol{z}}$ has order greater than $p^{m_0 - j}$, since some z_i is not divisible by p^j, and therefore $|\alpha_i^{z_i}| > p^{n_i - j} \ge p^{m_0 - j}$, yielding a contradiction. □

Lemma 4 not only tells us how to find independent elements, it gives sufficient conditions to ensure that this is possible. This yields a remarkably simple algorithm to construct a basis from a generating set S.

Start with $\boldsymbol{\alpha}$ consisting of a single element of S with maximal order p^m. Every $\beta \in S$ then satisfies $|\beta| \le p^m = p^{m_0}$, and we may use Algorithm 1 to compute an independent $\gamma = \beta \boldsymbol{\alpha}^{-\boldsymbol{x}}$ for each β. We can then choose one with maximal order to extend our basis $\boldsymbol{\alpha}$ and continue in this fashion until we have a basis spanning the entire group generated by S.

Algorithm 4. Given a subset S of a finite abelian p-group, the following algorithm computes a basis $\boldsymbol{\alpha}$ for $G = \langle S \rangle$:

1. Set $\boldsymbol{\alpha} \leftarrow \emptyset$ and compute $h_i \leftarrow \log_p |\beta_i|$ for each $\beta_i \in S$.
2. If every $h_i = 0$, return $\boldsymbol{\alpha}$.
   Otherwise pick a maximal h_i, set $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} \circ \beta_i$, and then $\beta_i \leftarrow 1_G$ and $h_i \leftarrow 0$.
3. For each $h_i > 0$:
   a. Compute $(\boldsymbol{x}, h) \leftarrow \mathrm{DL}^*(\boldsymbol{\alpha}, \beta_i)$ using Algorithm 3.
   b. Set $\beta_i \leftarrow \beta_i \boldsymbol{\alpha}^{-\boldsymbol{x}}$ and $h_i \leftarrow h$.
4. Go to step 2.

After step 1 we have (trivially) $\beta_i \perp \boldsymbol{\alpha}$ for all β_i, and after step 2 we must have $h_i \le \min(\log_p |\alpha_i|)$ for all h_i. By Lemma 4, these statements remain true after step 3, and at every step the algorithm ensures that $h_i = \log_p |\beta_i|$ and $\langle \boldsymbol{\alpha}, S \rangle = G$. If every $h_i = 0$, then $\langle S \rangle$ is trivial and $\boldsymbol{\alpha}$ is a basis for G. Some nonzero h_i is set to zero each time step 2 is executed, so this eventually happens. Note that when an element β_i is appended to $\boldsymbol{\alpha}$, its order p^{h_i} is known, as desired.

Proposition 2. *Given a set S that generates an abelian group G of size p^n, exponent p^m, and rank r, there is a generic algorithm to compute a basis for G using*

$$T_{\mathrm{B}}(S) \le c\Big(\frac{r \lg(m+1)}{\lg\lg(m+2)} \lg|G| + \frac{n}{r} p^{(r-1)/2} + \big(|S| - r\big) T_{\mathrm{DL}}(G)\Big)$$

group operations, where $T_{\mathrm{DL}}(G)$ is as in Proposition 1 *and c is an absolute constant independent of S and G.*

Proof. We apply Algorithm 4, setting $t = 1$ in Algorithm 3. We assume that a table of all nontrivial p^jth powers of each element of $\boldsymbol{\alpha}$ is maintained throughout, for a total cost of at most $2 \lg |G|$ group operations (this table can also be made available to Algorithm 3, avoiding the need for any precomputation). For each $\beta \in S$, computing $|\beta|$ in step 1 requires less than $2m \lg p$ group operations. The cost of all the exponentiations in step 3b related to β is bounded by $2 \lg |G|$ (if we consider $\boldsymbol{x} = \mathrm{DL}(\boldsymbol{\alpha}, \beta)$ for the initial value of β relative to the final basis $\boldsymbol{\alpha}$, each base-p digit of x is "cleared" in step 3b at most once). The total cost of all steps other than 3a is thus $O(|S| \lg |G|)$ group operations, which is bounded by the sum of the first and last terms in the bound for $T_B(S)$, for a suitable constant c.

We now consider the cost of step 3a for those $\beta = \beta_i \in S$ for which h_i is never chosen in step 2, meaning β_i is never appended to $\boldsymbol{\alpha}$. There are exactly $|S| - r$ such β. For each base case that succeeds in some computation $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$, the order of β is reduced by a factor of p in step 3b, so there are at most m successful base cases relevant to β in the entire execution of Algorithm 4. Ignoring the cost of reaching the first base case, and failed base cases, the successful part of all the $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$ computations involving β corresponds to a single computation $\mathrm{DL}(\boldsymbol{\alpha}, \beta)$ with respect to the final basis $\boldsymbol{\alpha}$ for G, which we bound by $T_{\mathrm{DL}}(G)$.

When computing $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$, reaching the first base case involves exponentiating β (and precomputing $\boldsymbol{\alpha}(j,k)$, but this was addressed above). The order of β is bounded by the order of the most recently added component α_i of $\boldsymbol{\alpha}$, hence the total cost of all the initial exponentiations of β is at most $\sum_{i=1}^{r} 2 \lg |\alpha_i| = 2 \lg |G|$, which is bounded by a constant factor of $T_{\mathrm{DL}}(G)$. Summing over the $|S| - r$ different values of β yields the term $(|S| - r) T_{\mathrm{DL}}(G)$ in the bound for $T_{\mathrm{B}}(S)$.

It remains to consider the cost of step 3a for the elements $\beta_1, \ldots, \beta_r$ that are at some point appended to $\boldsymbol{\alpha}$. With $n_i = \log_p |\alpha_i|$, we have $m = n_1 \ge \cdots \ge n_r \ge 1$.

Define $m_i = n_i - n_{i+1}$ for $1 \le i \le r-1$, and let $f(x) = x \lg(x+1)/\lg\lg(x+2)$. Excluding base cases, the successful part of all computations $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta_i)$ may be bounded, as in Proposition 1, by a constant factor of

$$\sum_{i=1}^{r-1}(r-i)f(m_i)\lg p \;\le\; (r-1)f(m)\lg p \;<\; \frac{r\lg(m+1)}{\lg\lg(m+2)}\lg|G|, \tag{16}$$

where we have used $\sum f(m_i) \le f(m)$ for positive integers m_i with $\sum m_i \le m$. As above, the total cost of reaching the first base case in the computations $\mathrm{DL}^*(\boldsymbol{\alpha}, \beta_i)$ for β_i is at most $2\lg|G|$, which may be incorporated into (16), yielding the first term of the bound for $T_{\mathrm{B}}(S)$.

Finally, we consider the cost of the base cases occurring for $\beta_1, \ldots, \beta_r$. In the ith iteration of step 3a, there are $r-i$ elements β_i which have yet to be appended to $\boldsymbol{\alpha}$, and exactly one base case fails for each of these. Thus we may bound the cost of all failed base cases by a constant factor of

$$\sum_{i=1}^{r-1}(r-i)p^{i/2} < \frac{p^{(r+1)/2}}{(\sqrt{p}-1)^2} < 12p^{(r-1)/2}. \tag{17}$$

For the successful base cases, let r_j be the rank of the subgroup of p^jth powers in G, as in Proposition 1, so that $n = r_0 + \cdots + r_{m-1}$. For each r_j we obtain a sum of the form (17), and note that, as in Corollary 1, we may bound the cost to within a constant factor by assuming the r_j are all approximately equal to r. In this case we have $m = \lceil n/r \rceil$, yielding the term $(n/r)p^{(r-1)/2}$ in the bound for $T_{\mathrm{B}}(S)$, which also covers the failed base cases, for a suitable choice of c. □

If we are given a set S of independent elements, Proposition 2 implies that we can typically verify that S is a basis for $G = \langle S \rangle$ more quickly than we can compute discrete logarithms in G. In fact this is true whenever $|S| = r$, even if the elements of S are not independent. More generally, we have the following corollary.

Corollary 2. *Given a generating set S for a finite abelian p-group G of rank r, with $|S| = r + O(1)$, there is a generic algorithm to compute a basis for G using*

$$T_{\mathrm{B}}(G) \;=\; O\left(\lg^{2+\epsilon}|G|\right) + O\big(T_{\mathrm{DL}}(G)\big) \;=\; O(|G|^{1/2})$$

group operations.

When a generating set is not available, or when $|S| \gg r$, we may instead use a probabilistic algorithm to construct a basis from randomly sampled elements of G. If r is known (or bounded), Corollary 2 can be applied to a randomly generated subset $S \subset G$ of size $r+t$ to obtain a generic Monte Carlo algorithm that is correct[10] with probability at least $1 - p^{-t}$. This follows from the lemma below, whose proof can be found in [21, Eq. 2] and also [1, Lem. 4].

Lemma 5. *Let G be a finite abelian p-group of rank r, and let S be a sample of $s \ge r$ independent and uniformly distributed random elements of G.*

Then S generates G with probability $\displaystyle\prod_{j=s-r+1}^{s}\left(1-p^{-j}\right) > 1 - p^{r-s}$.

[10]This algorithm always outputs a basis for a subgroup H of G, but it may be that $H < G$.

In general, we do not know the rank of G, *a priori*. Indeed, determining r may be a reason for computing a basis. In this situation we could apply Algorithm 4 to progressively larger randomly generated sets S until $|S| > r + t$, where r is the rank of $\langle S \rangle$ and t is a constant. However, a more efficient approach is to simply select random $\beta \in G$, using the black box or via Lemma 6 below, and attempt to use Lemma 4 to extend the current basis.

This eliminates the loop in step 3 of Algorithm 4, but we must now address the situation where Lemma 4 fails to apply ($|\beta| > p^m$ or $|\gamma| > p^{m_0}$). It may happen that the basis we have constructed cannot be extended to a basis for G, and in this case we need to backtrack. Fortunately, this is easy to detect (and correct) and has negligible impact on the expected running time.

Algorithm 5. Given a randomized black box for a finite abelian p-group G and $t \in \mathbb{Z}_{>0}$, the following algorithm computes a basis $\boldsymbol{\alpha}$ for a subgroup H of G, where $H = G$ with probability at least $1 - p^{-t}$:

1. Set $s \leftarrow 0$. Pick a random $\alpha_1 \in G$ and set $\boldsymbol{\alpha} \leftarrow (\alpha_1)$.
2. If $s = t$, then return $\boldsymbol{\alpha}$.
3. Pick a random $\beta \in G$ and compute $(\boldsymbol{x}, h) \leftarrow \mathrm{DL}^*(\boldsymbol{\alpha}, \beta)$.
4. If $h = 0$, then increment s and go to step 2; otherwise set $\gamma \leftarrow \beta \boldsymbol{\alpha}^{-\boldsymbol{x}}$.
5. For each α_i with $|\alpha_i| < |\gamma|$, remove α_i from $\boldsymbol{\alpha}$ and set $s \leftarrow 0$.
6. Set $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} \circ \gamma$ and go to step 2.

The correctness of Algorithm 5 depends on an easy corollary to Lemma 4. If we let the (possibly empty) vector $\boldsymbol{\alpha}'$ consist of those components of $\boldsymbol{\alpha}$ that satisfy $|\alpha_i| \geq |\gamma|$, then $\gamma \perp \boldsymbol{\alpha}'$ (the proof is the same). It follows that after step 6, $\boldsymbol{\alpha}$ is a basis for the subgroup it generates (this is obviously also true after step 1). When the algorithm terminates, it has found t (independent, uniformly distributed) random elements $\beta \in G$ that lie in $H = \langle \boldsymbol{\alpha} \rangle$. If H is a proper subgroup of G, it must be smaller by a factor of at least p; the probability that t random elements $\beta \in G$ all happen to lie in H is then at most p^{-t}.

Proposition 3. *Given a randomized black box for a finite abelian p-group G of rank r, exponent p^m, and size p^n, and $t \in \mathbb{Z}_{>0}$, there is a probabilistic generic algorithm that computes a basis for a subgroup H of G using an expected*

$$T_B^*(G) \;\leq\; c\Big(\frac{r \lg(m+1)}{\lg\lg(m+2)} \lg|G| + \frac{n}{r} p^{(r-1)/2}\Big) + t\,T_{\mathrm{DL}}(G) \;=\; O(|G|^{1/2})$$

group operations, such that $H = G$ with probability at least $1 - p^{-t}$. The absolute constant c is independent of both t and G.

Proof. We apply Algorithm 5. If it never backtracks (removes elements from $\boldsymbol{\alpha}$ in step 5), the final basis $\boldsymbol{\alpha}$ is obtained from the first r random elements, and then t discrete logarithms are computed using this basis. In this case, the bound $T_B^*(G)$ follows from an argument similar to that used in the proof of Proposition 2, with $|S| = r$ (and a better constant factor). We will show that the expected cost of Algorithm 5 is within a constant factor of the cost arising in this ideal scenario.

Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)$ be the final basis output by Algorithm 5, and note that $|\alpha_1| \geq |\alpha_2| \geq \cdots \geq |\alpha_r|$. As the computation proceeds, for each k from 1 to r, there is a stage k where $\boldsymbol{\alpha}_{k-1} = (\alpha_1, \ldots, \alpha_{k-1})$ is a (possibly empty) prefix of the final basis, and the algorithm is in the process of determining α_k. This may involve extending and then backtracking to the prefix $\boldsymbol{\alpha}_{k-1}$ (several times, perhaps), but

once α_k is determined, we have the prefix $\boldsymbol{\alpha}_k$ and transition to stage $k+1$. If no backtracking occurs, the algorithm completes stage 1 after step 1, and a single computation of $\mathrm{DL}^*(\boldsymbol{\alpha}_k, \beta)$ is required for each stage $k > 1$. From Proposition 1, the cost of this computation may be bounded by

$$S_k = c_1 \sum_{i=1}^{k-1} \frac{\lg(n_i+1)}{\lg\lg(n_i+2)} \lg|\alpha_i| + c_1 \sum_{j=0}^{m-1} p^{r_j/2} = A_k + B_k,$$

where c_1 is a constant, $n_i = \log_p |\alpha_i|$, and the ranks $r_j \le k-1$ are as in Proposition 1. Let A_k and B_k denote the two sums in S_k, including the factor c_1.

We now consider the probability that the computation $\mathrm{DL}^*(\boldsymbol{\alpha}_{k-1}, \beta)$ completes stage k. Let $\boldsymbol{z}$ be the discrete logarithm of β relative to the final basis $\boldsymbol{\alpha}$. Provided that z_k is not divisible by p, when h is computed in step 3 we will have $h = n_k$, and compute $\gamma = \alpha_k$ in step 4, since no subsequent computation can yield an independent element of order greater than n_k (since $n_j \le n_k$ for $j > k$). Thus for each random $\beta \in G$ processed during stage k, the probability that we do not complete stage k is at most $1/p$ (this is true for any extension of $\boldsymbol{\alpha}_{k-1}$ arising during stage k). Conditioning on w, the number of random $\beta \in G$ processed during stage k, the expected cost of stage k may be bounded by a sum of the form

$$T_k \le (1-p^{-1})(A_k+B_k) + p^{-2}((1+2)A_k + (1+p^{1/2})B_k) + \dots,$$

$$T_k \le \left(\frac{p-1}{p} + \sum_{w=2}^{\infty} \binom{w+1}{2} p^{-w}\right) A_k + \left(\frac{p-1}{p} + \sum_{w=2}^{\infty} b p^{-w/2}\right) B_k, \tag{18}$$

where $b = 1/(\sqrt{p}-1)$. We have assumed here, as a worst case, that after processing each β the current basis is extended by a γ that maximizes the cost of subsequent discrete logarithm computations. For each increment in w we suppose that $|\langle\boldsymbol{\alpha}\rangle|$ increases by a factor of $|\langle\boldsymbol{\alpha}_{k-1}\rangle|$ (in fact, it increases by at most a factor of $|\alpha_{k-1}|$) and that every r_j increases by 1.

The second sum in (18) is a geometric series, bounded by $b/(p-\sqrt{p}) < 5$. Summation by parts yields the identity

$$\sum_{w=1}^{\infty} \binom{w+1}{2} p^{-w} = \frac{p^2}{(p-1)^3},$$

allowing us to bound the first sum in (18) by 4. Hence $T_k \le c_2 S_k$ for a constant $c_2 < 6$, and the bound on $T_B^*(G)$ follows. The correctness probability was addressed above, and clearly $c = c_1 c_2$ is independent of t and G. □

In practice, the constant c in Proposition 3 is quite small and $T_B^*(G) \approx t T_{\mathrm{DL}}(G)$, even when $p = 2$ (the worst case, as far as the constant factors are concerned). When $T_{\mathrm{DL}}(G)$ is dominated by $p^{r/2}$, the constant t can be improved to $\sqrt{t}$ using a baby-steps giant-steps approach, as discussed in Section 6.

If we are given a bound M satisfying $M \le |G| < pM$ (perhaps $M = |G|$), we can easily convert Algorithm 5 from a Monte Carlo algorithm to a Las Vegas algorithm by replacing the test "$s = t$" in step 2 with "$|\langle\boldsymbol{\alpha}\rangle| \ge M$" (note that $|\langle\boldsymbol{\alpha}\rangle| = \prod |\alpha_i|$).

We now give a method to construct uniformly random elements of G from a generating set S. This is useful in general and allows us to apply Algorithm 5 to a generating set S, which may be faster than using Algorithm 4 when $|S| \gg r$.

Lemma 6. *Given a generating set S for a finite abelian group G with exponent $p^m = E$, and $t \in \mathbb{Z}_{>0}$, there is a generic algorithm to compute t independent, uniformly random elements of G using*

$$T_R(S,t) \leq (3\lg E)|S| + 2t\left(\frac{\lceil \lg(E+1)\rceil}{\lceil \lg\lg(E+2)\rceil} + 1\right)|S|$$

group operations and storage for at most $t + \lg E$ group elements.

Proof. We represent S as a vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_s)$. To construct random elements $\beta_1, \ldots, \beta_t$, first set each β_j to 1_G. Then, for each γ_i, compute $|\gamma_i| = p^{n_i}$, select t uniformly random integers $z_{i,j} \in [0, p^{n_i})$, and set $\beta_j \leftarrow \beta_j \gamma_i^{z_{i,j}}$ for each j. We assume all the $z_{i,j}$ are chosen independently.

The cost of computing $|\gamma_i|$ is at most $2m \lg p = 2 \lg E$ group operations. By Yao's Theorem, the cost of t exponentiations of the common base γ_i is at most

$$\lg E + ct\left(\frac{\lceil \lg(E+1)\rceil}{\lceil \lg\lg(E+2)\rceil}\right)$$

group operations, where $c \leq 2$. Accounting for the multiplication by β_j and summing over the γ_i yields the bound $T_R(S,t)$. We only need to store the β_j and at most $\lg E$ powers of a single γ_i at each step (we don't count the size of the input set S, since we only access one element of S at a time).

Clearly the β_j are independent; we must show that each is uniformly distributed over G. Let $H = \mathbb{Z}/p^{n_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p^{n_s}\mathbb{Z}$. The map $\varphi : H \to G$ that sends $\boldsymbol{z}$ to $\boldsymbol{\gamma}^{\boldsymbol{z}}$ is a surjective group homomorphism, and we have $\beta_j = \varphi(\boldsymbol{z})$, where $\boldsymbol{z} = (z_{1,j}, z_{2,j} \ldots, z_{s,j})$ is uniformly distributed over H. As each coset of $\ker\varphi$ has the same size, it follows that β_j is uniformly distributed over $G \cong H/\ker\varphi$. □

In practice, we may wish to generate random elements "on demand", without knowing t. We can generate random elements in small batches of size $t \approx \lg\lg(E+2)$ to effectively achieve the same result. If S is reasonably small, the first term of $T_R(S,t)$ may be treated as a precomputation and need not be repeated.

Provided that $|S| = O(|G|^{1/2-\epsilon})$, we may apply Lemma 6 and Proposition 3 to compute a basis for $G = \langle S\rangle$, with high probability, using $O(|G|^{1/2})$ group operations. By contrast, Algorithm 4 uses $O(|S||G|^{1/2})$ group operations when S is large, as does the algorithm of Buchmann and Schmidt [8]. However, we note that both of these algorithms are (or can be made) deterministic.

5. Constructing a basis in the general case

We now suppose that G is an arbitrary finite abelian group. If we know the exponent of G, call it $\lambda(G)$, and its factorization into prime powers, we can easily reduce the computation of a basis for G to the case already considered. In fact, it suffices to know any reasonably small multiple N of $\lambda(G)$, including $N = |G|$. Factoring N does not require any group operations, and it is, in any event, a much easier problem than computing $\lambda(G)$ in a generic group; hence we ignore this cost.[11]

As shown in the author's thesis, $\lambda(G)$ can be computed using $o(|G|^{1/2})$ group operations [26]. This bound is strictly dominated by the worst-case complexity of both the algorithms presented in the previous section, allowing us to extend our

[11] We have subexponential-time probabilistic algorithms for factoring versus exponential lower bounds for computing the group exponent with a probabilistic generic algorithm [3]. Most deterministic factoring algorithms are already faster than the $\Omega(N^{1/3})$ lower bound of [26, Thm. 2.3].

complexity bounds for abelian p-groups to the general case. The basic facts needed for the reduction are given by the following lemma.

Lemma 7. *Let G be a finite abelian group and let N be a multiple of $\lambda(G)$. Let $p_1, \dots, p_k$ be distinct primes dividing N, and let G_{p_i} be the Sylow p_i-subgroup of G.*

(i) *Given a generating set S for G, one can compute generating sets $S_1, \dots, S_k$ for $G_{p_1}, \dots, G_{p_k}$, each of size $|S|$, using $O\left(|S| \lg^{1+\epsilon} N\right)$ group operations.*

(ii) *Given a uniformly distributed random $\beta \in G$, one can compute elements $\beta_1, \dots, \beta_k$ uniformly distributed over the groups $G_{p_1}, \dots, G_{p_k}$ (respectively), using $O\left(\lg^{1+\epsilon} N\right)$ group operations.*

Proof. Let N_i be the largest divisor of N relatively prime to p_i. Given $\beta \in S$, or a random $\beta \in G$, we compute $\beta_1 = \beta^{N_1}, \dots, \beta_k = \beta^{N_k}$ with either Algorithm 7.3 or Algorithm 7.4 of [26], using $O(\lg^{1+\epsilon} N)$ group operations.[12]

The map $\phi_i : G \to G_{p_i}$ sending β to β^{N_i} is a surjective group homomorphism, invertible on $G_{p_i} \subset G$. Thus if S generates G, then $S_i = \phi_i(S)$ generates G_{p_i}, which proves (i). If β is uniformly distributed over G, then $\phi_i(\beta)$ is uniformly distributed over G_{p_i}, proving (ii). □

We now extend Propositions 2 and 3 to arbitrary finite abelian groups. To simplify the statement of results, we suppose that N is equal to either $\lambda(G)$ or $|G|$.

Proposition 4. *Let G be a finite abelian group whose nontrivial Sylow subgroups are $G_{p_1}, \dots, G_{p_k}$, and suppose that the exponent (resp. order) of G is given. Let S be a generating set for G, with S_i as in Lemma 7.*

(i) *There is a generic algorithm to compute a basis for G which uses*

$$O\big(|S| \lg^{1+\epsilon} |G|\big) \; + \; \sum T_{\mathrm{B}}(S_i)$$

group operations, where $T_{\mathrm{B}}(S_i)$ is bounded as in Proposition 2.

(ii) *Given a randomized black box for G, there is a Monte Carlo (resp. Las Vegas) generic algorithm to compute a basis for G using an expected*

$$O\big(\lg^{2+\epsilon} |G|\big) \; + \; \sum T_B^*(G_{p_i}) = O(|G|^{1/2})$$

group operations, where $T_B^*(G_{p_i})$ is bounded as in Proposition 3.

Proof. (i) is immediate from Lemma 7 and Proposition 2. (ii) follows similarly from Proposition 3 and the comments following, using the bound

$$\sum |G_{p_i}|^{1/2} \; \le \; \frac{3}{2} \prod |G_{p_i}|^{1/2} \; = \; \frac{3}{2} |G|^{1/2}$$

from Lemma 9. □

Corollary 3. *Given a randomized black box for a finite abelian group G, there is a probabilistic algorithm to compute a basis for G using $O(|G|^{1/2})$ group operations.*

Proof. Algorithm 8.1 of [26] computes $N = \lambda(G)$ (with high probability) and uses $o(\sqrt{N})$ group operations, assuming Algorithms 5.1 and 5.2 of [26] are used for order computations. The corollary then follows from (ii) of Proposition 4. □

[12] Algorithm 7.3 is due to Celler and Leedham-Green [10].

If we are given a generating set S with $|S| = O(|G|^{1/2-\epsilon})$, we may apply Lemma 6 to obtain an analogous corollary.

The space required by the algorithms of Proposition 4 and Corollary 3 can be made quite small, polynomial in $\lg |G|$, using algorithms based on Pollard's rho method to handle the base cases of the discrete logarithm computations and applying the search used in Algorithm 5.1 of [26]. If this is done, the complexity bound for computing $\lambda(G)$ increases to $O(N^{1/2})$ (but will typically be better than this).

It is not necessary to use a particularly fast algorithm to compute $\lambda(G)$ in order to prove Corollary 3; any $O(N^{1/2})$ algorithm suffices. However, the time to compute a basis for G is often much less than $|G|^{1/2}$ group operations, as the worst case may arise rarely in practice. Applying Algorithms 5.1 and 5.2 of [26] yields considerable improvement in many cases.[13]

These comments are especially relevant when one only wishes to compute a basis for a particular Sylow p-subgroup H of G (perhaps as a prelude to extracting pth roots in G). Once we have computed $\lambda(G)$, we can compute a basis for any of G's Sylow subgroups with a running time that typically depends only on the size and shape of the subgroup of interest, not on G. The following proposition follows immediately from Lemma 7 and Proposition 3.

Proposition 5. *Let H be a Sylow p-subgroup of a finite abelian group G. Given a multiple N of the exponent of G and a randomized black box for G, there is a probabilistic generic algorithm to compute a basis for H using*

$$O\big(r \lg^{1+\epsilon} N\big) + T_B^*(H) = O(r \lg^{1+\epsilon} N + |H|^{1/2})$$

group operations, where r is the rank of H.

6. Performance results

We tested the new algorithms on abelian p-groups of various sizes and shapes in order to assess their performance. As in previous sections, G is an abelian group of size p^n, exponent p^m, and rank r, whose shape is given by a partition of n into r parts, with largest part m.

Here we present results for $p = 2$, as this permits the greatest variation in the other parameters, and also because the Sylow 2-subgroup is of particular interest in many applications. Results for other small primes are similar. When p is large, the results are not as interesting: n, r, and m are all necessarily small, and the computation is dominated by the discrete logarithms computed in the base cases, whose $\Theta(p^{r/2})$ performance is well understood.

Our tests in p-groups used a black box which represents each cyclic factor of G using integers mod p^{n_i}. This is a convenient but arbitrary choice. Identical results are obtained for any black box implementation, since the algorithms are generic. Our performance metric counts group operations (multiplications and inversions) and does not depend on the speed of the black box or the computing platform.[14]

To compute discrete logarithms in the base cases, we used Shanks' baby-steps giant-steps algorithm [22] extended to handle products of cyclic groups. Rather than the lexicographic ordering used by Algorithm 9.3 of [26], we instead compute

[13]For example, Teske reports computing a basis for the ideal class group G of $\mathbb{Q}[\sqrt{D}]$, with $D = -4(10^{30}+1)$, using $243,207,644 \approx 7.1|G|^{1/2}$ group operations [28]. In [26], a basis for G is computed using $250,277 \approx 2.4|G|^{1/3}$ group operations.

[14]Thus the performance results reported here are not impacted by Moore's law.

TABLE 1. Group operations to compute $DL(\boldsymbol{\alpha}, \beta)$ in $G \cong \left(\mathbb{Z}/2^{n/r}\mathbb{Z}\right)^r$.

$n\backslash r$	**1**	**2**	**4**	**8**	**16**	**32**
32	113	89	76	94	669	97936
64	261	204	172	194	853	163750
128	591	455	380	370	1501	197518
256	1268	1021	833	760	2065	328839
512	2718	2165	1770	1607	3760	395187
1024	5949	3931	3755	4601	5745	657965

a Gray code [13] when enumerating steps, always using one group operation per step (this is especially useful for small p, saving up to a factor of 2). A more significant optimization available with Shanks' method is the ability to perform k discrete logarithms in a group of size N using $2\sqrt{kN}$ (rather than $2k\sqrt{N}$) group operations by storing $\sqrt{kN}$ baby steps in a lookup table and then taking $\sqrt{N/k}$ giant steps as each of the k discrete logarithms is computed.[15]

This optimization is useful in Algorithms 1 and 3, even for a single discrete logarithm computation, as there may be many base cases in the same subgroup. It is even more useful in the context of Algorithms 4 and 5, as several calls to Algorithm 3 may use the same basis. In the bound for $T_{\text{DL}}(G)$ in Corollary 1, this effectively replaces the factor n/r by $\sqrt{n/r}$. When the rank-dependent terms in $T_{\text{DL}}(G)$ dominate sufficiently, the bounds in Propositions 2 and 3 can be improved by replacing $|S| - r$ with $\sqrt{|S| - r|}$ and t with $\sqrt{t}$ (respectively).

Table 1 lists group operation counts for Algorithm 1 when computing discrete logarithms in 2-groups of rectangular shape, corresponding to partitions of n into r parts, all of size $m = n/r$. Each entry is an average over 100 computations of $\text{DL}(\boldsymbol{\alpha}, \beta)$ for a random $\beta \in G$. Precomputation was optimized for a single discrete logarithm (repeated for each β), and these costs are included in Table 1. Reusing precomputed values can improve performance significantly over the figures given here, particularly when additional space is used, as in [27].

Algorithm 1 used the parameter $t = \lfloor (\lg n - 1)/r \rfloor$ in these tests, which was near optimal in most cases. The optimal choice of w is slightly less than that used in the proof of Proposition 1, as the average size of the exponents is smaller than the bound used there. For each entry in Table 1, if one computes the bound on $T_{\text{DL}}(G)$ given by Proposition 1, we find the constant c close to 1 in most cases (never more than 1.5). In the first four columns of Table 1, the counts are dominated by the exponent-dependent terms of $T_{\text{DL}}(G)$, explaining the initially decreasing costs as r increases for a fixed value of n (r is larger, but $m = n/r$ is smaller).

Table 2 compares the performance of Algorithm 1 to Teske's generalization of the Pohlig–Hellman algorithm on groups of order 2^{256} with a variety of different shapes. The baby-steps giant-steps optimization mentioned above is also applicable to Teske's algorithm (with even greater benefit), and we applied this optimization to both algorithms. The figures in Table 2 reflect averages for 100 computations of $DL(\boldsymbol{\alpha}, \beta)$ for random $\beta \in \langle \boldsymbol{\alpha} \rangle$.

[15] One uses $\sqrt{kN/2}$ baby steps to optimize the expected case, assuming $\beta \in \langle \boldsymbol{\alpha} \rangle$.

TABLE 2. Computing discrete logarithms in groups of order 2^{256}.

Group	Structure	Pohlig–Hellman–Teske	Algorithm 1
G_1	256	32862	1268
G_2	$128 \cdot 64 \cdot 32 \cdot 16 \cdot 8 \cdot 4 \cdot 2 \cdot 1^2$	8736	1095
G_3	16^{16}	2065	1036
G_4	$26 \cdot 22 \cdot 21 \cdot 20 \cdots 3 \cdot 2 \cdot 1$	17610	6647
G_5	$128 \cdot 32^2 \cdot 8^4 \cdot 2^8 \cdot 1^{16}$	534953	84047
G_6	$226 \cdot 1^{30}$	1075172	81942

The notation $a \cdot b^c$ indicates the group $\mathbb{Z}/2^a\mathbb{Z} \times (\mathbb{Z}/2^b\mathbb{Z})^c$.

Two advantages of Algorithm 1 are apparent in Table 2. In the first two rows, the complexity is dominated by m, and Algorithm 1 has a nearly linear dependence on m, versus a quadratic dependence in the algorithms of Pohlig–Hellman and Teske. In the last two rows, the complexity is dominated by $p^{r/2}$. Algorithm 1 computes just one base case in a subgroup of size p^r, due to the shapes of the groups, while Teske's algorithm computes m base case in a subgroup of size p^r (using $O(m^{1/2}p^{r/2})$ group operations, thanks to the baby-steps giant-steps optimization).

Table 3 presents performance results for Algorithms 4 and 5 when used to construct a basis for four of the groups listed in Table 2. The group operation counts are averages for 100 tests. The first four rows list results for Algorithm 4 when given a random generating set S of size $r + t$. The case $t = 0$ is of interest because it covers the situation where S is itself a basis, hence it may function as a basis verification procedure. The costs in this case are comparable to the cost of computing a single discrete logarithm in the group generated by S (this improves for $p > 2$).

The last four rows of Table 3 give corresponding results for Algorithm 5 using a randomized black box. In the first row for Algorithm 5, the algorithm is given the order of the group and runs as a Las Vegas algorithm, terminating only when it has found a basis for the entire group. In the remaining rows, Algorithm 5 is used as a Monte Carlo algorithm, correct with probability at least $1 - p^{-t}$.

TABLE 3. Computing a basis for groups of order 2^{256}.

	t	G_2	G_3	G_4	G_5
Algorithm 4	0	897	1739	9231	169633
	20	27077	15383	50528	406102
	40	45741	24946	71752	586501
	80	82921	44337	111451	788065
Algorithm 5	-	12727	2770	49219	372876
	20	27725	15027	68362	494345
	40	44137	26066	79950	587645
	80	76054	40843	109257	936478

Acknowledgments

The author would like to thank David Harvey for his extensive feedback on an early draft of this paper, and the referee for improving the proof of Lemma 6.

7. Appendix

The inequality below is elementary and surely known. Lacking a suitable reference, we provide a short proof here.

Lemma 8. *For any real number $a > 1$ there is a constant $c \le a/e^{1+\ln\ln a}$ such that for all real numbers $x_1, \dots, x_n \ge a$ (and any n),*

$$\sum x_i \;\le\; c \prod x_i.$$

Proof. We assume $x_1 \le \cdots \le x_n$. If we fix $\sum x_i$, we can only decrease $\prod x_i$ by supposing $x_{n-1} = a$, since if $x_{n-1} = a + \delta$, we have

$$x_{n-1}x_n = ax_n + \delta x_n \le ax_n + \delta a = a(x_n + \delta).$$

We now assume $x_1 = \cdots = x_{n-1} = a$ and $x_n = a + \delta$ with $\delta \ge 0$. Since

$$f(\delta) \;=\; \sum x_i / \prod x_i \;=\; \frac{(n-1)a + \delta}{a^{n-1}\delta}$$

is a decreasing function of δ, we maximize $\sum x_i / \prod x_i$ by assuming $x_n = a$ as well.

Thus it suffices to consider the case $\sum x_i / \prod x_i = na/a^n$, and we now view $g(n) = na/a^n$ as a function of a real variable n, which is maximized by $n = 1/\ln a$. Therefore, we may bound $\sum x_i / \prod x_i$ by $(1/\ln a)/a^{1/\ln a} = a/e^{1+\ln\ln a}$, and the lemma follows. □

The bound on c given in the lemma is not necessarily tight, since n must be an integer. If we note that $g(n) = na/a^n$ is increasing for $n < 1/\ln a$ and decreasing for $n > 1/\ln a$, it follows that the best possible c is

$$c = \min\left(g\left(\left\lfloor \frac{1}{\ln a} \right\rfloor\right), g\left(\left\lceil \frac{1}{\ln a} \right\rceil\right)\right). \tag{19}$$

Applying (19) with $a = \sqrt{2}$, we obtain the following lemma.

Lemma 9. *For any integers $x_1, \dots, x_n > 1$ we have $\sum \sqrt{x_i} \;\le\; \frac{3}{2} \prod \sqrt{x_i}$.*

References

1. Vincenzo Acciaro, *The probability of generating some common families of finite groups*, Utilitas Mathematica **49** (1996), 243–254. MR1396305 (97c:20106)
2. Leonard M. Adleman, Kenneth Manders, and Gary L. Miller, *On taking roots in finite fields*, Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 175–178. MR0502224 (58:19339)
3. László Babai and Robert Beals, *A polynomial-time theory of black-box groups. I*, Groups St. Andrews 1997 in Bath, I, London Mathematical Society Lecture Notes Series, vol. 260, Cambridge University Press, 1999, pp. 30–64. MR1676609 (2000h:20089)
4. Daniel J. Bernstein, *Faster square roots in annoying finite fields*, `http://cr.yp.to/papers/sqroot.pdf`, 2001.
5. ———, *Pippenger's exponentiation algorithm*, `http://cr.yp.to/papers/pippenger.pdf`, 2001.
6. Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David B. Wilson, *Fast exponentiation with precomputation*, Advances in Cryptology–EUROCRYPT '92, Lecture Notes in Computer Science, vol. 658, Springer-Verlag, 1992, pp. 200–207.

7. Johannes Buchmann, Michael J. Jacobson, Jr., and Edlyn Teske, *On some computational problems in finite abelian groups*, Mathematics of Computation **66** (1997), 1663–1687. MR1432126 (98a:11185)
8. Johannes Buchmann and Arthur Schmidt, *Computing the structure of a finite abelian group*, Mathematics of Computation **74** (2005), 2017–2026. MR2164109 (2006c:20108)
9. Johannes Buchmann and Ulrich Vollmer, *Binary quadratic forms: An algorithmic approach*, Algorithms and Computations in Mathematics, vol. 20, Springer, 2007. MR2300780 (2008b:11046)
10. Frank Celler and C. R. Leedham-Green, *Calculating the order of an invertible matrix*, Groups and Computation II, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 28, American Mathematical Society, 1997, pp. 55–60. MR1444130 (98g:20001)
11. Henri Cohen, *A course in computational algebraic number theory*, Springer, 1996. MR1228206 (94i:11105)
12. Daniel M. Gordon, *A survey of fast exponentiation methods*, Journal of Algorithms **27** (1998), 129–146. MR1613189 (99g:94014)
13. Donald E. Knuth, *The art of computer programming*, vol. IV, fascicle 2: Generating all tuples and permutations, Addison-Wesley, 2005. MR2251595 (2007f:68004b)
14. Chae Hoon Lim and Pil Joong Lee, *More flexible exponentiation with precomputation*, Advances in Cryptology–CRYPTO '94, Lecture Notes in Computer Science, vol. 839, Springer, 1994, pp. 95–107. MR1316405 (95k:94020)
15. Kevin S. McCurley, *The discrete logarithm problem*, Cryptography and Computational Number Theory (C. Pomerance, ed.), Proceedings of Symposia in Applied Mathematics, vol. 42, American Mathematical Society, 1990, p. 4974. MR1095551 (92d:11133)
16. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997, revised reprint. MR1412797 (99g:94015)
17. Andrew Odlyzko, *Discrete logarithms: The past and the future*, Designs, Codes, and Cryptography **19** (2000), 129–145. MR1759614
18. Nicholas Pippenger, *On the evaluation of powers and related problems (preliminary version)*, 17th Annual Symposium on Foundations of Computer Science, IEEE, 1976, pp. 258–263. MR0483702 (58:3682)
19. Stephen C. Pohlig and Martin E. Hellman, *An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance*, IEEE Transactions on Information Theory **24** (1978), 106–110. MR0484737 (58:4617)
20. John M. Pollard, *Monte Carlo methods for index computations mod p*, Mathematics of Computation **32** (1978), 918–924. MR0491431 (58:10684)
21. Carl Pomerance, *The expected number of random elements to generate a finite abelian group*, Periodica Mathematica Hungarica **43** (2001), 191–198. MR1830576 (2002h:20101)
22. Donald Shanks, *Class number, a theory of factorization and genera*, Analytic Number Theory, Proceedings of Symposia on Pure Mathematics, vol. 20, American Mathematical Society, 1971, pp. 415–440. MR0316385 (47:4932)
23. ———, *Five number-theoretic algorithms*, Proceedings of the 2nd Manitoba Conference on Numerical Mathematics, 1972, pp. 51–70. MR0371855 (51:8072)
24. Victor Shoup, *Lower bounds for discrete logarithms and related problems*, Advances in Cryptology–EUROCRYPT '97, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, revised version, pp. 256–266. MR1603068 (98j:94023)
25. ———, *A computational introduction to number theory and algebra*, Cambridge University Press, 2005. MR2151586 (2006g:11003)
26. Andrew V. Sutherland, *Order computations in generic groups*, Ph.D. thesis, MIT, 2007, `http://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf`.
27. ———, *Extracting roots in finite abelian groups*, 2008, in preparation.
28. Edlyn Teske, *A space efficient algorithm for group structure computation*, Mathematics of Computation **67** (1998), 1637–1663. MR1474658 (99a:11146)
29. ———, *Speeding up Pollard's rho method for computing discrete logarithms*, Algorithmic Number Theory Symposium–ANTS III, Lecture Notes in Computer Science, vol. 1423, Springer-Verlag, 1998, pp. 541–554. MR1726100 (2000j:11199)

30. _____, *The Pohlig-Hellman method generalized for group structure computation*, Journal of Symbolic Computation **27** (1999), 521–534. MR1701092 (2000f:20090)
31. Alberto Tonelli, *Bemerkung über die Auflösung quadratischer Congruenzen*, Göttinger Nachrichten (1891), 344–346.

Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

E-mail address: drew@math.mit.edu