Efficient recovering of operation tables of black box groups and rings

Jens Zumbrägel, Gérard Maze and Joachim Rosenthal
Mathematics Institute
University of Zurich
Winterthurerstrasse 190
CH - 8057 Zurich, Switzerland
www.math.unizh.ch/aa

***Abstract*— People have been studying the following problem: Given a finite set S with a hidden (black box) binary operation $* : S \times S \to S$ which might come from a group law, and suppose you have access to an oracle that you can ask for the operation $x * y$ of single pairs $(x, y) \in S^2$ you choose. What is the minimal number of queries to the oracle until the whole binary operation is recovered, i.e. you know $x * y$ for all $x, y \in S$?**

This problem can trivially be solved by using $|S|^2$ queries to the oracle, so the question arises under which circumstances you can succeed with a significantly smaller number of queries.

In this presentation we give a lower bound on the number of queries needed for general binary operations. On the other hand, we present algorithms solving this problem by using $|S|$ queries, provided that $*$ is an abelian group operation. We also investigate black box rings and give lower und upper bounds for the number of queries needed to solve product recovering in this case.

I. Introduction

There is a considerable literature on algebraic objects whose operations are described by a 'black box'. There are different motivations for studying such objects.

In computational group theory, *black box groups* became an important and frequently used tool. They were introduced by Babai and Szemerédi [BS84] in order to study algorithms for matrix groups. In a black box group elements are encoded as (not necessarily unique) bitstrings and there are oracles (the black box) providing multiplication and inversion of the encoded group elements as well as recognition of the identity element—but often the isomorphy class and even the order of the underlying group is unknown. The research is ongoing and a collection of some algorithms for black box groups can be found in Seress' monograph [Ser03, Ch. 2]. One major question in black box group research is the recognition problem which asks whether a given black box group is isomorphic to a fixed finite group like S_n or $SL_n(\mathbb{F}_q)$ and possibly to provide an explicit description of such an isomorphism, see e.g. [BLGN$^+$03], [BP00], [KS01]. The constructive recognition problem in the case of abelian groups has been investigated by Buchmann, Jacobson, Schmidt and Teske [BJT97], [BS05].

Also in cryptography the use of black box groups and fields proved themselves useful when analyzing the hardness of the discrete logarithm problem, e.g. Shoup obtained this way lower bounds for generic algorithms [Sho97]. Black box fields of prime order were used by Boneh and Lipton [BL96] when studying the discrete logarithm problem in a presence of a Diffie-Hellman oracle. Note that the isomorphy class of the underlying objects are known in these cases.

The basic question we are interested in this paper is, given a black box group or black box ring, how many calls to the oracle are necessary to recover the whole operation tables. To illustrate the importance of this question we mention its relevance for estimating the information content of an algebraic operation table and for designing practical compression algorithms for these tables. Furthermore, a black box object might be a crucial device in a symmetric cryptosystem and one wishes to analyze the cost to describe this black box completely.

We shall be interested in the problem of recovering the hidden operation by using a minimal number of queries to the oracle. In algorithm analysis we neglect here the remaining computational costs, i.e. we assume unlimited computational and storage power but limited access to the oracle.

When investigating the product recovering problem it is natural to assume unique encoding of group elements, since for the black box operation to depend only on the underlying group and not on some encoding arbitrariness this is necessary. However we include some remarks

Authors were supported in part by Swiss National Science Foundation Grant no. 107887.

978-1-4244-2571-6/08/$25.00 ©2008 IEEE

and comments concerning the general case of nonunique encoding.

The organization of this paper is as follows. In Section II, which forms the main part, we consider the case of one binary black box operation. After a formalization of the problem we are able to prove lower bounds for the general case in Subsection II-B and for some special cases in Subsection II-C. Afterwards we present some upper bounds and give the corresponding algorithms for the case of abelian groups in Subsection II-D. Here the lower and upper bounds are quite close together.

Finally in Section III we consider algebraic structures with two binary operations. We deal with the situation where we have a ring with known addition but unknown multiplication.

II. One binary operation

We define a black box with one binary operation in the most general way:

Defintion 1: A *black box groupoid* is a given finite set S together with a binary operation $* : S \times S \to S$ which is accessible by an oracle. The oracle can be asked for the multiplication $x * y$ of single pairs $(x, y) \in S^2$.

The set S can be thought of as a set of bitstrings and the binary operation $*$ is the black box we only have limited access to.

Given a black box groupoid, we are interested in the problem of recovering the hidden operation $*$ by using a minimal number of queries to the oracle. We also assume that some information on the operation $*$ is available, i.e. a set $\mathcal{X}$ of possible binary operations $* : S \times S \to S$ is given. For example, if we know that $*$ is a group operation, then

$$\mathcal{X}_{\text{Groups}} = \{* : S \times S \to S \mid (S, *) \text{ is a group}\}.$$

Another example is the situation where we know that $(S, *)$ is isomorphic to a particular groupoid $(G, \cdot)$. In this case

$$\mathcal{X}_G = \{* : S \times S \to S \mid \text{there is an isomorphism } f : (S, *) \to (G, \cdot)\}.$$

Algorithms solving the product recovering problem must specify the appropriate set $\mathcal{X}$.

Remark 2: When only the existence of an epimorphism $f : (S, *) \to (G, \cdot)$ is known, then we may have nonunique encoding of group(oid) elements. This is usually the case in black box group literature, where there is also an oracle for testing whether $f(x) = 1$ holds. However, in general we cannot exploit the algebraic structure of G to recover exactly $*$. Instead we can hope to find a subset $\tilde{S} \subseteq S$ such that $f|_{\tilde{S}} : \tilde{S} \to G$ is bijective and to find $\tilde{*} : \tilde{S} \times \tilde{S} \to \tilde{S}$ such that $f(a * b) = f(a \tilde{*} b)$ for all $a, b \in S$.

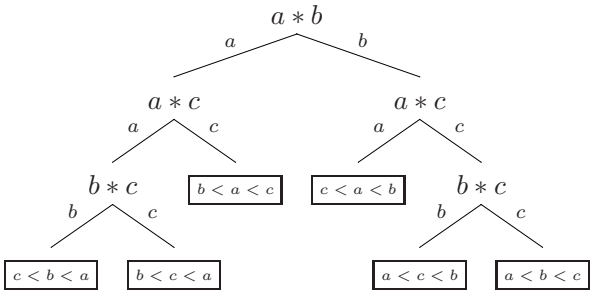

Fig. 1. A query-algorithm for a totally ordered set $\{a, b, c\}$

A. Query-algorithms

We model query-algorithms as certain labeled trees with the nodes corresponding to queries to the oracle and the edges corresponding to its possible answers:

Defintion 3: A *query-algorithm* with respect to a set $\mathcal{X}$ of binary operations $* : S \times S \to S$ on a set S is a rooted tree T with labels such that

- any node v of T which is not a leaf is labeled with '$x * y$' where $x, y \in S$ (to be thought of as a query), leaves are unlabeled,
- the branches to the children of v are labeled with 'z' with elements $z \in S$ (to be thought of as possible answers), such that different branches have different labels.

Furthermore we require *completeness of answers* in the following sense. For every possible binary operation $* \in \mathcal{X}$ there exists a corresponding path $(v_0, \ldots, v_k)$ from the root v_0 to a leaf v_k such that if v_i is labeled with '$x_i * y_i$' then the branch (v_i, v_{i+1}) is labeled with 'z_i' where $z_i = x_i * y_i$, for $0 \le i < k$.

The leaf $L(*) =: v_k$ is then uniquely determined by this property, so that there is a well-defined map

$$L : \mathcal{X} \to \{\text{leaves of } T\}.$$

The query-algorithm T is said to *solve product-recovering* if this map L is bijective, i.e. there is a one-one correspondence between the leaves of T and the operations $* \in \mathcal{X}$.

Example 4: Let $\mathcal{X}$ be the set of all binary operations $*$ on a three-element-set $S = \{a, b, c\}$ such that $(S, *)$ is isomorphic to the semigroup $(\{0, 1, 2\}, \max)$. Thus, there is an unknown total ordering of the elements of S, and the problem of product-recovering is equivalent to find back this ordering.

A query-algorithm solving product-recovering is shown in Fig. 1. Every leaf v is labeled with the ordering which corresponds to the binary operation $*_v = L^{-1}(v)$.

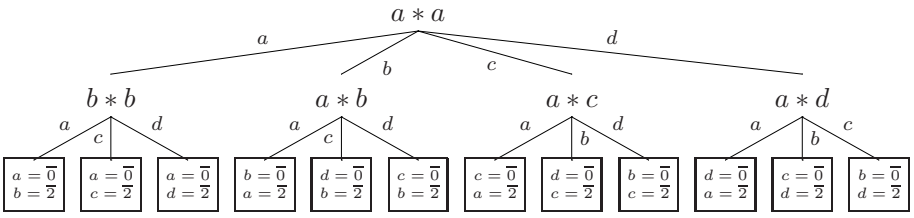

Fig. 2. A query-algorithm for the group $\mathbb{Z}_4 = \{\overline{0}, \overline{1}, \overline{2}, \overline{3}\}$

Example 5: Let $(G, \cdot) = (\mathbb{Z}_4, +)$ be the cyclic group of order 4 and let $S = \{a, b, c, d\}$. Fig. 2 shows a query-algorithm T solving product-recovering for this group, i.e. with respect to $\mathcal{X}_G$.

The leaves v are labeled with a shortened presentation of the binary operation $*_v = L^{-1}(v)$. Note that the elements $\overline{1}$ and $\overline{3}$ of $\mathbb{Z}_4$ are exchangeable, so we do not have to specify their corresponding elements in S.

Since T has 12 leaves we have $|\mathcal{X}_G| = 12 = \frac{4!}{|\operatorname{Aut}(\mathbb{Z}_4)|}$, which also follows from Lemma 7 below.

B. Lower bounds

The next lemma establishes a general lower bound.

Lemma 6: Let T be any query-algorithm which solves product-recovering with respect to a set of $\mathcal{X}$ of binary operations on a set S, and let N be its number of queries to the oracle. Assuming a uniform distribution on $\mathcal{X}$ we have for the expectation

$$E(N) \geq \log_{|S|} |\mathcal{X}|.$$

(We say T needs at least $\log_{|S|} |\mathcal{X}|$ queries *on average*.)

Proof: $E(N)$ is the average height of all leaves of the tree T. Now any node of T has at most $|S|$ children and T has exactly $|\mathcal{X}|$ leaves. This yields the result. ■

Lemma 7: For any groupoid $(G, \cdot)$ with $|G| = n$ we have

$$|\mathcal{X}_G| = \frac{n!}{|\operatorname{Aut}(G, \cdot)|}.$$

Proof: Without loss of generality we may assume that $S = G$ as sets.

Consider the set X of all binary operations $* : G \times G \to G$ and the group action

$$Sym(G) \times X \to X, \quad (\varphi, *) \mapsto *^{\varphi},$$

where $*^{\varphi}$ is defined by $x *^{\varphi} y = \varphi^{-1}(\varphi(x) * \varphi(y))$ for all $x, y \in G$, so that $\varphi : (G, *^{\varphi}) \to (G, *)$ is a groupoid isomorphism.

Now under this group action, the operation $\cdot \in X$, coming from the known groupoid $(G, \cdot)$, has exactly $\mathcal{X}_G$ as orbit and $\operatorname{Aut}(G, \cdot)$ as stabilizer group. Thus the lemma follows from the orbit-stabilizer theorem. ■

Now if $G_1, \ldots, G_m$ are pairwise non-isomorphic groupoids (e.g. the family of all abelian groups of a given size), then the $\mathcal{X}_{G_i}$ are pairwise disjoint. Let $\mathcal{X} := \mathcal{X}_{G_1} \cup \cdots \cup \mathcal{X}_{G_m}$, then

$$|\mathcal{X}| = n! \left(\sum_{i=1}^{m} \frac{1}{|\operatorname{Aut}(G_i, \cdot)|} \right).$$

C. Special cases

1) $\max$*-semigroups:* Assume that $(S, *)$ is isomorphic to the semigroup $(G, \cdot) = (\{0, 1, \ldots, n-1\}, \max)$ (see Example 4 for the case $n = 3$).

Corollary 8: Any query-algorithm which solves product-recovering with respect to $\mathcal{X}_G$ needs at least

$$\log_2(n!) \geq n \log_2 n - \frac{n}{\ln 2} + \frac{\log_2 n}{2}$$

queries to the oracle on average.

Proof: Since $|\operatorname{Aut}(G, \cdot)| = 1$ we have $|\mathcal{X}_G| = n!$ by Lemma 7. Now in this semigroup any node of a query-algorithm has at most 2 children, so by the same argument given in the proof of Lemma 6 we see that at least $\log_2(n!)$ queries on average are needed. Finally the stated inequality is a consequence of $n! \geq \left(\frac{n}{e}\right)^n \sqrt{n}$, coming from Stirling's formula. ■

Note that solving product-recovering reduces to the well-studied problem of sorting an n-element set, where the queries to the oracle correspond to comparisons of elements. There are several sorting algorithms (e.g. merge sort) which use $O(n \ln n)$ comparisons in the worst case.

2) Abelian groups: Now assume that $(S, *) \cong (G, \cdot)$ is an abelian group with n elements (see Example 5).

Corollary 9: Suppose that $(G, \cdot)$ is generated by r elements. Any query-algorithm which solves product-recovering with respect to $\mathcal{X}_G$ needs at least

$$n - \frac{n}{\ln n} + \frac{1}{2} - r$$

queries to the oracle on average.

Proof: Note that any endomorphism of G is determined by its image on its r generators. Hence $|\operatorname{Aut}(G)| \leq |\operatorname{End}(G)| \leq n^r$. On the other hand we

have $n! \geq \left(\frac{n}{e}\right)^n \sqrt{n}$ from Stirling's formula, so that $\log_n(n!) \geq n - \frac{n}{\ln n} + \frac{1}{2}$. Now the result follows from Lemma 6 and Lemma 7 ∎

Note that any abelian group of size n can be generated by at most $\log_2 n$ elements, so that one can achieve $r \leq \log_2 n$ in general. Of course, if $(G, \cdot)$ is cyclic, one can set $r = 1$.

D. Upper bounds for abelian groups

We give an upper bound for the worst-case number of queries needed to solve product-recovering in the case of abelian groups and present the corresponding algorithm in the proof.

Proposition 10: Let S be a set of size n and

$$\mathcal{X}_{\text{Ab}} = \{* : S \times S \to S \mid (S, *) \text{ is an abelian group}\}.$$

Then there is a query-algorithm which solves product-recovering with respect to $\mathcal{X}_{\text{Ab}}$ using at most n queries to the oracle for any $* \in \mathcal{X}$.

Proof: We write the query-algorithm as a list of instructions rather than as a tree, because this representation is more compact and readable. The algorithm is based on two basic subroutines.

1) Start by choosing some $a = a_1 \in S$ and apply the following algorithm.

Repeat computing $a_{k+1} = a_k * a$ for $k = 1, 2, 3, \ldots$ until $a_{k+1} = a$.

After execution, k is the order $\text{ord}(a)$ of a, and k queries to the oracle have been made. We further know that $a_0 := a_k$ is the identity element. Also, we deduce that $a_i * a_j = a_{i+j \mod k}$ for $0 \leq i, j < k$.

Hence if $S_a = \{a_0, a_1, \ldots, a_{k-1}\}$ is the subgroup generated by a, then $*$ is known on $S_a \times S_a$.

2) If $S_a \neq S$ choose some $b = b_1 \in S \setminus S_a$ and apply the following algorithm.

Repeat computing $b_k = b_{k-1} * b$ for $k = 2, 3, 4, \ldots$ until $b_k \in S_a$. For all $s \in S_a \setminus \{0\}$ and $0 < i < k$ compute $s * b^i$.

After execution we know $s * b^i$ for all $s \in S_a$ and all $0 \leq i < k$. Then for any $s, t \in S_a$ and $0 \leq i, j < k$ we have by commutativity

$$(s * b_i) * (t * b_j) = \begin{cases} (s * t) * b_{i+j} & \text{if } i + j < k, \\ (s * t * b_k) * b_{i+j-k} & \text{if } i + j \geq k. \end{cases}$$

This element is known, since we knew already $*$ on $S_a \times S_a$. It follows that $*$ is known on $S_{ab} \times S_{ab}$ where S_{ab} is the subgroup generated by S_a and b.

Let $m = |S_a|$. The number of queries to the oracle needed by the algorithm is

$$k - 1 + (m - 1)(k - 1) = m(k - 1) = mk - m.$$

Now $mk = |S_{ab}|$, so that $|S_{ab}| - |S_a|$ queries to the oracle have been used.

3) If $S_{ab} \neq S$ choose some $c \in S \setminus S_{ab}$ and repeat 2) with S_a replaced by S_{ab} and b replaced by c. After that $*$ is known on $S_{abc} \times S_{abc}$, the subgroup generated by S_{ab} and c, and $|S_{abc}| - |S_{ab}|$ queries to the oracle have been used, etc.

Writing $S_1, S_2, S_3, \ldots$ for $S_a, S_{ab}, S_{abc}, \ldots$ we finally reach r such that $S_r = S$. Then we have recovered the whole operation $*$ on $S \times S$ and we have used

$$|S_1| + (|S_2| - |S_1|) + \cdots + (|S_r| - |S_{r-1}|) = |S_r| = |S| = n$$

queries to the oracle in total. ∎

Example 11: Consider a black box group $(S, *)$ of size 11. Then we know that S is isomorphic to the cyclic group and $|\text{Aut}(S, *)| = 10$. By Lemma 6 and Lemma 7 we conclude that an algorithm solving product-recovering needs at least $\lceil \log_{11} 3991680 \rceil = 7$ queries to the oracle in the worst case.

Proposition 10 ensures the existence of an algorithm which needs 11 oracle-queries. In fact it is not hard to see that in the case of groups of prime order the last two queries in the above algorithm can be omitted, yielding an algorithm which uses 9 queries to the oracle.

Now a computer search among all possible product recovering algorithms has shown that the minimal number of oracle-queries an algorithm needs in the worst-case is 8. Such an algorithm can be outlined as follows:

1) choose some $a \in S$ and compute $a_{\text{sq}} = a * a$
2) if $a_{\text{sq}} \neq a$ let $a_1 = a$ and $a_2 = a_{\text{sq}}$, otherwise let $e = a$, choose some $a_1 \neq a$ and compute $a_2 = a * a$
3) compute $a_3 = a_2 * a_1$, $a_4 = a_3 * a_1$, $a_5 = a_4 * a_1$ and $a_7 = a_5 * a_2$
4) if $a_{\text{sq}} \neq a$ compute $e = a_7 * a_4$
5) choose three different elements b, c, d from the four-element-set $S \setminus \{e, a_1, a_2, a_3, a_4, a_5, a_7\}$ and compute $b * c$ and $b * d$

Remark 12: When we have nonunique encoding of group elements a variant of the above algorithm will solve product-recovering provided we are given a generating set and an oracle for recognizing the identity. However, to check whether b_k lies in S_a (as in the second subroutine) may be a costly operation.

III. Two binary operations

Suppose we are given a finite set S with *two* hidden binary operations $+$ and $*$, accessible via an oracle. If we

know that $(S,+,*)$ is a ring, then $(S,+)$ is an abelian group, so we can use Proposition 10 to recover the addition table. We now have a ring with known addition, but unknown multiplication. This section deals with that situation.

Defintion 13: A *black box groupoid with given addition* is a black box groupoid $(S,*)$ such that there is a known binary operation $+ : S \times S \to S$ on S, and the following distributive laws hold on S with respect to $+$ and $*$, i.e.

$$\left.\begin{array}{l} a*(b+c) = (a*b)+(a*c) \\ (a+b)*c = (a*c)+(b*c) \end{array}\right\} \text{ for all } a,b,c \in S.$$

In this case, all binary operations $* \in \mathcal{X}$ in question will satisfy the distributive laws above. Suppose, for example, we know that $(S,+,*)$ is isomorphic to some known ring $(R,+,\cdot)$. Then the set of possible operations we are dealing with is

$$\mathcal{X}_R := \{* : S \times S \to S \mid \text{there is an isomorphism } \varphi : (S,+,*) \to (R,+,\cdot)\}.$$

Its size is given in the next result.

Lemma 14: For any ring $(R,+,\cdot)$ we have

$$|\mathcal{X}_R| = \frac{|\operatorname{Aut}(R,+)|}{|\operatorname{Aut}(R,+,\cdot)|},$$

where $\operatorname{Aut}(R,+)$ are the additive group automorphisms and $\operatorname{Aut}(R,+,\cdot)$ are the ring automorphisms.

Proof: The arguments are the same as in the proof of Lemma 7. We identify $S = R$ as sets and consider the action of $\operatorname{Aut}(R,+)$ on the set X of all binary operations $* : R \times R \to R$. Then $\mathcal{X}_R$ is exactly the orbit of $\cdot \in X$ and $\operatorname{Aut}(R,+,\cdot)$ is its stabilizer group. ■

Now we specialize to the case when $(S,+,*) \cong (\mathbb{F}_q,+,\cdot)$ is a field of size $q = p^r$ with p prime.

Corollary 15: Any query-algorithm which solves product-recovering for a field of size $q = p^r$ with known addition needs at least

$$r - \log_q(4r)$$

queries to the oracle on average.

Proof: The automorphisms $\operatorname{Aut}(\mathbb{F}_q,+)$ are exactly the vector space automorphisms of the $\mathbb{F}_p$-vector space $(\mathbb{F}_p)^r$, so that

$$\begin{aligned} |\operatorname{Aut}(\mathbb{F}_q,+)| &= (q-1)(q-p)\cdots(q-p^{r-1}) \\ &= q^r\left(1-\frac{1}{p}\right)\left(1-\frac{1}{p^2}\right)\cdots\left(1-\frac{1}{p^r}\right), \end{aligned}$$

where $(1-\frac{1}{p})\cdots(1-\frac{1}{p^r}) > \prod_{i\geq 1}(1-\frac{1}{2^i}) > \prod_{i\geq 0}(\frac{1}{2})^{\frac{1}{2^i}} = \frac{1}{4}$. On the other hand, $|\operatorname{Aut}(\mathbb{F}_q,+,\cdot)| = |\operatorname{Aut}(\mathbb{F}_{p^r}/\mathbb{F}_p)| = [\mathbb{F}_{p^r} : \mathbb{F}_p] = r$, by basic Galois theory. Hence

$$\frac{|\operatorname{Aut}(\mathbb{F}_q,+)|}{|\operatorname{Aut}(\mathbb{F}_q,+,\cdot)|} \geq \frac{q^r}{4r}$$

and the result follows from Lemma 6 and Lemma 14. ■

We now give an upper bound for the number of queries needed to solve product-recovering for rings $(S,+,*)$ of size $|S| = n$ with given addition. For this it suffices to ask the oracle for all products $a*b$ of elements $a,b \in A$, where A is a generating set for the abelian group $(S,+)$. Then if $x,y \in S$, we can write $x = a_1 + \cdots + a_k$ and $y = b_1 + \cdots + b_l$ with $a_i, b_j \in A$ for all i,j, and thus

$$x*y = (a_1+\cdots+a_k)*(b_1+\cdots+b_l) = \sum_{i=1}^{k}\sum_{j=1}^{l} a_i * b_j;$$

now, since all $a_i * b_j$ are known and the addition is known, we also know $x*y$.

Because $(S,+)$ can be generated by at most $\log_2 n$ elements, we thus have $(\log_2 n)^2$ as an upper bound. Together with Proposition 10 this proves:

Proposition 16: If $(S,+,*)$ is a ring of size n, then there is a query-algorithm which solves product-recovering for both operations $+$ and $*$ with at most

$$n + (\log_2 n)^2$$

queries to the oracle in the worst case.

References

[BJT97] J. Buchmann, M. J. Jacobson, Jr., and E. Teske, *On some computational problems in finite abelian groups*, Math. Comp. **66** (1997), no. 220, 1663–1687.

[BL96] D. Boneh and R. J. Lipton, *Algorithms for black-box fields and their application to cryptography (extended abstract)*, Advances in cryptology—CRYPTO '96 (Santa Barbara, CA), Lecture Notes in Comput. Sci., vol. 1109, Springer, Berlin, 1996, pp. 283–297.

[BLGN+03] R. Beals, C. R. Leedham-Green, A. C. Niemeyer, C. E. Praeger, and Á. Seress, *A black-box group algorithm for recognizing finite symmetric and alternating groups. I*, Trans. Amer. Math. Soc. **355** (2003), no. 5, 2097–2113 (electronic).

[BP00] S. Bratus and I. Pak, *Fast constructive recognition of a black box group isomorphic to S_n or A_n using Goldbach's conjecture*, J. Symbolic Comput. **29** (2000), no. 1, 33–57.

[BS84] L. Babai and E. Szemerédi, *On the complexity of matrix group problems i*, 25th Annual Symposium on Foundations of Computer Science (1984), 229–240.

[BS05] J. Buchmann and A. Schmidt, *Computing the structure of a finite abelian group*, Math. Comp. **74** (2005), no. 252, 2017–2026 (electronic).

[KS01] W. M. Kantor and Á. Seress, *Black box classical groups*, Mem. Amer. Math. Soc. **149** (2001), no. 708, viii+168.

[Ser03] Á. Seress, *Permutation group algorithms*, Cambridge Tracts in Mathematics, vol. 152, Cambridge University Press, Cambridge, 2003.

[Sho97] V. Shoup, *Lower bounds for discrete logarithms and related problems*, Advances in cryptology—EUROCRYPT '97 (Konstanz), Lecture Notes in Comput. Sci., vol. 1233, Springer, Berlin, 1997, pp. 256–266.