Factoring using multiplicative relations modulo n: a subexponential algorithm inspired by the index calculus

Katherine E. Stange[1,*]

[1]Department of Mathematics, University of Colorado Boulder

Received: October 27, 2024 | Revised: October 27, 2024 | Accepted: October 27, 2024

Abstract *We demonstrate that a modification of the classical index calculus algorithm can be used to factor integers. More generally, we reduce the factoring problem to finding an overdetermined system of multiplicative relations in any factor base modulo n, where n is the integer whose factorization is sought. The algorithm has subexponential runtime* $\exp(O(\sqrt{\log n \log\log n}))$ *(or* $\exp(O((\log n)^{1/3}(\log\log n)^{2/3}))$ *with the addition of a number field sieve), but requires a rational linear algebra phase, which is more intensive than the linear algebra phase of the classical index calculus algorithm. The algorithm is certainly slower than the best known factoring algorithms, but is perhaps somewhat notable for its simplicity and its similarity to the index calculus.*

Keywords: Integer factoring, index calculus

2010 Mathematics Subject Classification: 11Y05

1 INTRODUCTION

The index calculus (whose roots date to Kraitchik [11] and which was rediscovered and developed more recently by Adleman [1] and Western and Miller [24], among others) computes the discrete logarithm of a residue h modulo p with respect to a base g by collecting relations of the form

$$g^x = \prod_{i=1}^{b} p_i^{e_i} \pmod{p},$$

in terms of a factor base of primes p_i. Choosing subexponentially many primes, it takes subexponential time to find as many relations as there are primes. This leads to a linear algebra system, whose solution gives the discrete logarithms of the primes, which can then be used to find the discrete logarithm of h.

It has long been observed that factoring algorithms and discrete logarithm algorithms have many similarities. In the present note, we demonstrate a factoring algorithm that is especially closely inspired by the index calculus algorithm. The fundamental idea is easy to state: we attempt to use the index calculus algorithm modulo an integer n with unknown factorization (instead of a prime p). That is, with respect to a factor base of primes, collect relations of the form

$$g^x = \prod_{i=1}^{b} p_i^{e_i} \pmod{n}.$$

In this case, the exponent relations live modulo the Euler totient $\varphi(n)$, or, more precisely, modulo the order of the chosen base g. We do not know this order, and in fact its discovery would typically lead to a factorisation of n. Although the discrete logarithms of a factor base need not exist in this situation, multiple relations will tend to lead to a linear system that is overdetermined and has no solution, unless working modulo the order of g. This leads to a linear algebra method to extract the order of g.

It turns out that more generally, any method of finding an overdetermined system of multiplicative relations between elements of a factor base modulo n will lead to a method of factorization. We give a simple heuristic reduction from factoring to finding such a system of multiplicative relations:

Theorem 1 (Introductory form of Theorem 4)**.** *Let n be an integer, and let b be a function of n. Let* $\mathcal{B}$ *be a factor base of b residues modulo n. Suppose an oracle provides random multiplicative relations of size* $\leq O(\log n)$ *amongst the residues* $\mathcal{B}$*. Then there is a Las Vegas algorithm to determine the multiplicative order of residues*

*Corresponding Author: kstange@math.colorado.edu

modulo n (and hence factor n), whose runtime is polynomial in b and $\log n$*, and which requires at most* $O(b)$ *calls to the oracle. Furthermore, under Hypothesis 3 (Section 3), the probability of success approaches* $1 - 1/\zeta(c+1)$ *where a total of* $b + c$ *calls are made to the oracle.*

The notation ζ denotes the Riemann zeta function. Although the algorithm may fail (it may return a nontrivial multiple of the order of g, or return 0), a simple and informal implementation using SageMath [19] indicates the algorithm succeeds with very high probability. Hypothesis 3 concerns the probability that a random selection of elements of a lattice will generate that lattice, or, more precisely, that the same is true for a restriction to a one-dimensional subspace. The probability $1 - 1/\zeta(k)$ is more familiar as the probability that k random integers have no non-trivial common factor.

In fact, taking $c = 1$ should in general suffice, using methods of Ekerå; see Section 5.

The runtime of the new index-calculus-like algorithm is slower than that of the textbook index calculus, because the linear algebra step involves arithmetic over $\mathbb{Q}$. Its runtime is $\exp(O((\log n)^{1/2}(\log\log n)^{1/2}))$. The textbook index calculus described above has been improved by the addition of the number field sieve in the precomputation (relation-finding) phase [8]. The methods of [8, Section 3.1] can be adapated to find relations modulo n, which, when combined with Theorem 4, leads to a version of the present algorithm which runs in time $\exp(O((\log n)^{1/3}(\log\log n)^{2/3}))$. We do not pursue this in greater detail here.

The literature for factoring includes a panoply of approaches. As far as index-calculus-style approaches to factoring, another family of approaches uses an index calculus within the class group of a quadratic field [12, 14, 21]. The analysis of Section 3 has some commonalities with the Hafner-McCurley rigorous class group algorithm [9], and their methods may be applicable to making the present algorithm rigorous. Although the present algorithm is closely related to the index calculus and other relation-finding approaches such as the quadratic and number fields sieves, the author has been unable to find this particular variation in the literature. For background on the index calculus, the number field sieve and factoring algorithms in general, see [4].

ACCOMPANYING CODE

A toy implementation is available at

`https://github.com/katestange/index-factor`

ACKNOWLEDGEMENTS

Thank you especially to Martin Ekerå, who pointed out that knowing only a *multiple* of the order was sufficient, based on his recent work (see Section 5). Thank you to Carl Pomerance and Sam Wagstaff for sharing their expertise in the factoring literature, and to Sam Wagstaff and Renate Scheidler for helpful comments on an earlier draft. Thank you to Karthekeyan Chandrasekaran for help with a reference. And finally, the author is grateful to the students in her Fall 2022 Introduction to Coding Theory and Cryptography at the University of Colorado Boulder for their enthusiasm and engagement, which led, indirectly, to this paper.

2 THE ALGORITHM

The algorithm is actually one for finding the multiplicative order of g modulo n. It is well-known that this leads to a factorization of n. For example, Shor's quantum factoring algorithm determines this order r for random values of g until it finds an order r which is even and for which $g^{r/2} \neq -1 \pmod{n}$ [22]. In this case, $\gcd(g^{r/2} \pm 1, n)$ should be a non-trivial factor. More generally, including odd orders, see [5].

By the *period problem* for an integer n, we shall mean the problem of computing the multiplicative order of a given residue modulo n. The *factorization problem* for an integer n is the problem of giving its unique prime factorization over the integers. It is known that these are equivalent.

Theorem 2 ([15, Theorem 4])**.** *Under the Extended Riemann Hypothesis, the period problem and the factorization problem are polynomial-time equivalent.*

The Extended Riemann Hypothesis is used to guarantee the existence of a small residue which generates the multiplicative group; in practice, choosing a residue at random is likely to result in a generator very quickly.

We now describe the algorithm. Let n be an odd positive integer, and let g be a residue modulo n. Let $\mathcal{B}$ be a factor base consisting of the primes $p_1, \ldots, p_b$ less than or equal to a bound B, chosen depending upon n (so $b = \#\mathcal{B}$).

For random integers $0 < x_j < n$, compute $g^{x_j} \pmod{n}$ and attempt to factor the resulting smallest positive

residue in the factor base. For each relation obtained in this way, being of the form

$$g^{x_j} = \prod_{i=1}^{b} p_i^{f_{j,i}} \pmod{n},$$

store the vector $\mathbf{f}_j = (f_{j,i})_{i=1}^{b} \in \mathbb{Z}^b$ and the value x_j. Collect these vectors until we have more than b vectors; say $\mathbf{f}_1, \ldots, \mathbf{f}_{b+c}$. (Taking $c = 10$ or so should generally suffice, but this is a parameter of the algorithm.)

We should now find integer relations between the vectors $\mathbf{f}_j$. These relations can be found as elements of the basis of the right kernel of the $b \times (b + c)$ matrix whose columns are the relations, working over $\mathbb{Q}$. Using gcd computations, scale c of these basis elements to have integral entries with no common factor, and call them $\mathbf{b}_1, \ldots, \mathbf{b}_c \in \mathbb{Z}^{b+c}$. In other words,

$$\sum_{j=1}^{b+c} (\mathbf{b}_t)_j \mathbf{f}_j = \mathbf{0},$$

for $t = 1, \ldots, c$. Collect the values

$$\alpha_t := \sum_{j=1}^{b+c} (\mathbf{b}_t)_j x_j.$$

We return the gcd of this set of values. We show below that this gcd is is with high probability the multiplicative order of g modulo n, and otherwise an integer multiple of this multiplicative order. The algorithm is given more formally in Algorithm 2.1.

Algorithm 2.1: Computing the multiplicative order of g modulo n.

Input : A positive integer n, and a positive integer $g < n$.
Output The multiplicative order of g modulo n.
:

Inititialization phase:

Select a suitable $B \in \mathbb{N}$, and let $\mathcal{B} = \{p_1, p_2, \ldots, p_b\}$ be the *factor base* consisting of all primes less than B.
Select a suitable $c \in \mathbb{N}$ (typically $c \sim 10$ is fine).

Relation finding phase:

Let $j = 1$.
while $j < b + c$ **do**
    Choose an integer x randomly in the range $[1, \ldots, n]$. (If x has been drawn previously, draw again.)
    Compute the smallest positive residue of g^x modulo n.
    Attempt to factor the residue of g^x in terms of the factor base.
    if *g^x is factored, say* $g^x = \prod_{i=1}^{b} p_i^{f_i} \pmod{n}$ **then**
        Set $\mathbf{f}_j = (f_1, \ldots, f_b)$. Set $x_j = x$. Increment j.

Linear algebra phase:

Form the $b \times (b + c)$ integer matrix whose columns are the $\mathbf{f}_j$.
Compute independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_c \in \mathbb{Q}^{b+c}$ in the right kernel of this matrix (i.e., column combinations that vanish).

GCD computation phase:

Scale each basis element so that the entries are integers with no factor common to all entries.
for *t from* 1 *to* c **do**
    Let $\alpha_t = \sum_{j=1}^{b+c} (\mathbf{b}_t)_j x_j$.
Let G be the greatest common divisor of all the $\alpha_1, \ldots, \alpha_c$.
return G

2.1 CORRECTNESS

We now briefly demonstrate the claim that the result is a multiple of $\mathrm{ord}(g)$, the multiplicative order of g. For a more complete analysis of runtime and success probability, see the next section. When one has a relation

$$\sum_{j=1}^{b+c} (\mathbf{b}_t)_j \mathbf{f}_j = \mathbf{0},$$

this implies that

$$\prod_{j=1}^{b+c} (g^{x_j})^{(\mathbf{b}_t)_j} = 1 \pmod{n}.$$

In particular, this implies that

$$\alpha_t = \sum_{j=1}^{b+c} (\mathbf{b}_t)_j x_j = 0 \pmod{\mathrm{ord}(g)}.$$

Hence, the multiplicative order g modulo n must divide every α_t. Therefore the final gcd of the algorithm is a multiple of $\mathrm{ord}(g)$.

Remark 1. *Particularly when g has small order, it is likely that the individual discrete logarithms $L_g(p_i)$ don't all exist. However, the algorithm doesn't mind; the factorization is just a way to create multiplicative relations between values of g^x. If $L_g(p_i)$ and $L_g(p_j)$ don't exist, but $L_g(p_i p_j)$ does, then $L_g(p_i)$ will only appear with the same coefficient as $L_g(p_j)$ in the linear algebra, effectively reducing the number of variables and increasing the size of the kernel. But the correctness of the algorithm is not affected.*

3 FACTORING BY RELATIONS MODULO n

In order to analyse the algorithm's correctness and runtime, we will isolate the novel phase encompassing the linear algebra and gcd computations. In fact, this phase of the algorithm generalizes to give a heuristic reduction from factoring to finding multiplicative relations modulo n, which we will state in Theorem 4 below. This theorem will then imply that Algorithm 2.1 is correct and has a high probability of success, under Hypothesis 3 below. This hypothesis is known to hold in some cases, such as when the size of the factor base is much smaller than n, so that in those cases there is a reduction from factoring n to finding relations in such a factor base (under ERH).

If we have a factor base $\mathcal{B}$ of b residues $a_1, \ldots, a_b$ modulo n, then by a *multiplicative relation*, we shall mean a relationship

$$\prod_{i=1}^{b} a_i^{e_i} = 1$$

modulo n for $e_i \in \mathbb{Z}$. We give a name to the lattice of exponent vectors for all valid relations:

$$\Lambda_{\mathcal{B}} = \left\{ \mathbf{e} = (e_i)_{i=1}^{b} : \prod_{i=1}^{b} a_i^{e_i} = 1 \right\}.$$

This is a sublattice of $\mathbb{Z}^b$. If the residues generate $(\mathbb{Z}/n\mathbb{Z})^*$, then it will have covolume equal to $\varphi(n)$.

Let S_i be the i-th coordinate axis, i.e. vectors whose entries are non-zero only in the i-th position. The restriction $\Lambda_{\mathcal{B}}|_{S_i}$ of $\Lambda_{\mathcal{B}}$ to S_i has covolume equal to the multiplicative order of a_i, denoted $\mathrm{ord}(a_i)$. In particular, it is generated by a vector having i-th entry $\mathrm{ord}(a_i)$, and zeroes elsewhere. Equivalently, any generating set for $\Lambda_{\mathcal{B}}|_{S_i}$ will be of the form $\{d_j \epsilon_i\}_j$ where ϵ_i is the i-th standard basis vector, and $\gcd_j(d_j)$ is equal to $\mathrm{ord}(a_i)$.

The *size* of a relation $\mathbf{e}$ will be equal to the logarithm of the 1-norm $|\mathbf{e}|_1$. (Thus relation vectors whose entries are $< n$ have size $O(\log n)$.)

Hypothesis 3 (Hypothesis on sublattice generation)**.** *Let $n > 0$ tend to ∞, and $b = f(n)$ be a given function of n. Let $\mathcal{B}$ be a factor base of b invertible residues modulo n. Let $\Lambda'_{\mathcal{B}} \subseteq \Lambda_{\mathcal{B}}$ be a lattice generated by $b + c$ relations randomly chosen from amongst those in $\Lambda_{\mathcal{B}}$ of size $O(\log n)$. Then, asymptotically, the probability that $\Lambda'_{\mathcal{B}}|_{S_i} = \Lambda_{\mathcal{B}}|_{S_i}$ is equal to the probability that $c + 1$ random integers (in the sense of natural density) share no common factor, i.e. $1 - 1/\zeta(c+1)$ where ζ is the Riemann zeta function.*

If b and n satisfy the relationship $n \geq 8b^{\frac{b+1}{2}}$ as n tends to infinity, then taking $c = b + 1$, it is known that the probability has a positive lower bound [6, Theorem 1.1]. By contrast, to apply this result to Algorithm 2.1, we need the Hypothesis to hold when b is subexponential in $\log n$.

To provide some evidence for the Hypothesis, we give a heuristic argument. The index $[\Lambda_{\mathcal{B}}|_{S_i} : \Lambda'_{\mathcal{B}}|_{S_i}]$ is computed (for example in Algorithm 2.1) as follows: let K be the right kernel of the matrix M whose columns are the $b + c$ relations. It has dimension $\geq c$, and with high probability the dimension is exactly c (if the relations generate a sublattice of $\Lambda_{\mathcal{B}}$ of full rank; see [6] for a result that this is the case with constant probability once the relation size is large enough). Let M_i be the matrix obtained by deleting the i-th row of M. Then its right kernel $K_i \supseteq K$ (of dimension $c + 1$ with high probability) gives linear combinations of the original relations which are supported only on a_i, i.e. live in S_i. These generate $\Lambda'_{\mathcal{B}}|_{S_i}$, and each is an integer multiple of $\mathrm{ord}(a_i)\epsilon_i$, where ϵ_i is the i-th standard basis vector in $\mathbb{Z}^n$. Write α_t for these integers, $t = 1, \ldots, c + 1$.

Let us assume heuristically that the values α_t behave as random integers (in the sense of natural density). The probability that k random integers share no common factor is $1 - 1/\zeta(k)$, where ζ is the Riemann zeta function [17]. Therefore the probability that $\Lambda_{\mathcal{B}}|_{S_i} = \Lambda'_{\mathcal{B}}|_{S_i}$ would be $1 - 1/\zeta(c + 1)$.

As the convergence $\zeta(n) \to 1$ as $n \to \infty$ is exponential, we need only a polynomial number of extra relations to reduce the failure probability to be negligible. For example, we might expect $\Lambda_{\mathcal{B}}|_{S_i} = \Lambda'_{\mathcal{B}}|_{S_i}$ at least 99.9% of the time if $c \geq 9$.

It is worth noting that it is possible for the algorithm to return $\alpha_t = 0$. This occurs for elements of $K \subseteq K_i$, which forms a hyperplane of codimension 1; such occurrences are unlikely (by a cardinality argument similar to that in [6, Section 2.1]).

Theorem 4. *Let n be an integer, and let b be a function of n. Let $\mathcal{B}$ be a factor base of b invertible residues modulo n. Suppose an oracle provides random multiplicative relations of size $\leq O(\log n)$ amongst the residues $\mathcal{B}$. Then there is a Las Vegas algorithm to solve the period problem for modulus n, whose runtime is $\tilde{O}(b^9)\,\mathrm{poly}(\log n)$ and which requires at most $O(b)$ calls to the oracle. Furthermore, under Hypothesis 3, the probability of success approaches $1 - 1/\zeta(c + 1)$ where a total of $b + c$ calls are made to the oracle.*

We use the convention that a Las Vegas algorithm is one which may return $\perp$ (failure) with a finite probability less than 1.

Proof. The algorithm is as follows (similarly to the latter portions of Algorithm 2.1). Let $c > 0$ be an integer. Call the oracle $b + c$ times to obtain $b + c$ relations. Isolate variable a_1, rewriting the j-th relation for each j as

$$a_1^{e_{j,1}} = \prod_{i=2}^{b} a_i^{e_{j,i}}.$$

Then, by dimensional considerations, there are at least $c + 1$ vectors $(c_{k,j})_{j=1}^{b+c}$ for $k = 1, \ldots, c + 1$ which give linear combinations $\sum_{j=1}^{b+c} c_{k,j}\mathbf{e}_j$ which are supported only on the first entry. Equivalently, these satisfy

$$a_1^{\sum_{j=1}^{b+c} c_{k,j} e_{j,1}} = 1.$$

We can normalize these relations by scaling so the entries $c_{k,j}$ are integers with no common factor, by use of a gcd computation. Write $\alpha_k := \sum_{j=1}^{b+c} c_{k,j} e_{j,1} \in \mathbb{Z}$. Let g be the gcd of the set of exponents α_k, $k = 1, \ldots, c + 1$. Then g is a multiple of the multiplicative order of a_1 modulo n, say $g = h\,\mathrm{ord}(a_1)$.

By the discussion preceding the proof, $h = [\Lambda_{\mathcal{B}}|_{S_i} : \Lambda'_{\mathcal{B}}|_{S_i}]$. Under Hypothesis 3, the probability that $h = 1$ (and hence the algorithm succeeds) is $1 - 1/\zeta(c + 1)$. Failure is easily detected: finding the order is equivalent to factoring n, so if the obtained order does not lead to a factorization of n, we should return $\perp$.

We now take c to be $O(b)$, and consider the runtime. We must find a basis for the kernel of a $b \times (b + c)$ matrix, where $b + c = O(b)$. The entries are of size $O(\log n)$ (in the sense that they are integers $\leq n$). We can find the Smith Normal Form in time $O(b^\omega M(b \log n))$, where matrix multiplication takes $O(r^\omega)$ operations in dimension r and integer multiplication takes time $M(r)$ for r-bit integers [23]. Since $\omega < 3$ [2] and $M(r) = O(r \log r)$ [10], the Smith Normal Form computation takes time $O(b^4)\,\mathrm{poly}(\log n)$. With this, one can compute the kernel.

Thus the linear algebra takes time at most $O(b^4)\,\mathrm{poly}(\log n)$ resulting in vectors with entries of size $O(b^4)\,\mathrm{poly}(\log n)$ (using the loose approximation that runtime bounds output size; see also [20, Corollary 3.2d]). In the GCD phase, we must perform $O(b)$ gcd operations on integers of size at most $O(b^4)\,\mathrm{poly}(\log n)$; this has runtime $\tilde{O}(b^9)\,\mathrm{poly}(\log n)$. Overall, the runtime of the algorithm is $\tilde{O}(b^9)\,\mathrm{poly}(\log n)$. □

4 RUNTIME OF THE ALGORITHM

As usual, we set the notation

$$L_x(\alpha, \beta) = \exp((\beta + o(1))(\log x)^\alpha (\log\log x)^{1-\alpha}).$$

We will now show that the algorithm is of runtime $L_n(1/2, \beta)$ for some constant β, which can be improved by the use of many optimizations developed for the index calculus; see below. However, since this algorithm is of academic, not practical, interest, we will not devote time to optimizing the constant β.

The runtime of the linear algebra and gcd phase is given in Theorem 4. The relation finding phase is exactly as for the index calculus itself. If we use the standard notation u^u for the number of trials to find one smooth integer, where $u = \log n/\log b$, then, to find $b + c$ smooth integers by trial division (time $bM(b \log b) = b \log b \log\log b$ per trial division) the runtime is $u^u(b + c)b \log b \log\log b$ with trial division. Thus, optimizing, we take $b = L_n(1/2, 1/2)$ (for the standard analysis of such runtimes, see [4, Chapter 6] or [7, Chapter 15]), and obtain a runtime for the relation finding phase of $L_n(1/2, 1)$.

With this choice of b, the runtime of Theorem 4 becomes $L_n(1/2, 9)$. Thus, $\beta = 9$ in our implementation. If we use $c = 1$ as discussed in the next section, the GCD phase would be dispensed with, so that we get $L_n(1/2, 4)$.

5 IMPROVEMENTS

The work of Ekerå provides an algorithm to factor n given the order r of an invertible residue a_i. However, after this article was accepted, the author was made aware that the method of Ekerå works on multiples of r' of the order r, without any necessity to factor r'. This has an immediate effect on the algorithm presented in this paper, namely that it is no longer necessary to collect several α_t; one will suffice. Instead of Hypothesis 3, we would only need a heuristic on the probability that a certain number of randomly chosen relations in $\Lambda_{\mathcal{B}}$ will form a full-rank sublattice of $\Lambda_{\mathcal{B}}$. Once we have a full rank sublattice $\Lambda'_{\mathcal{B}}$, the intersection with S_i will be non-trivial, and obtain at least one non-trivial α_t, a multiple of the order r. The algorithm of Ekerå is also a Las Vegas algorithm, with a probability of failure that can be made to approach 0, under the assumption that the element a_i whose order we are finding is chosen at random from the multiplicative group. This is not strictly the case in the present application (since the a_i are the factor base, chosen with other considerations in mind). Nevertheless, in practice this would simplify the algorithm here.

Some other relevant implementation notes:

1. One might include -1 in the factor base.

2. One may use the linear sieve of Coppersmith, Odlyzko and Schroeppel [3]; this improves the relation-finding phase.

3. For the linear algebra phase, one might use an algorithm of runtime $O(b^3)\,\mathrm{poly}(\log n)$ due to Mulders and Storjohann [16].

4. One might use the elliptic curve method [13] to remove all prime factors below a bound before attempting this algorithm. Having many small factors will result in the order of g being smaller more often, which gives a higher failure rate in using the order to obtain the factorization.

5. We might test for the existence of a non-trivial kernel periodically as we generate relations, since if the order of g is small, the algorithm actually requires fewer relations.

6. In the same vein, we might use a single kernel element, when it is found, to obtain a multiple of $\mathrm{ord}(g)$ and then do further linear algebra modulo that modulus.

7. We may need to return to the relation-finding and add more relations if we do not find enough elements of the kernel which result in $\alpha_t \neq 0$.

8. Many implementation tricks for the index calculus may apply in this situation, see for example [18].

9. The number field sieve as in [8, Section 3.1] can be adapted to speed up the relation-finding phase.

6 EXAMPLE

Let us compute the order of $g = 43$ modulo $n = 62389$ and use this to factor n. We will use $B = 50$, resulting in a factor base of $b = 15$ primes $2 \leq p \leq 47$. We will need 25 relations (this assumes $c = 10$). With 188 smoothness tests, we find the relations:

$$43^{55571} = 2^3 \cdot 3^3 \cdot 7 \cdot 29,$$
$$43^{51344} = 5^4,$$
$$43^{1724} = 2 \cdot 5^3 \cdot 7 \cdot 23,$$
$$43^{9399} = 3 \cdot 13 \cdot 37,$$
$$43^{56136} = 2 \cdot 3 \cdot 11^2 \cdot 13,$$
$$43^{53393} = 5^4 \cdot 41,$$
$$43^{24567} = 2^4 \cdot 7 \cdot 23^2,$$
$$43^{2484} = 2 \cdot 3^2 \cdot 13 \cdot 37,$$
$$43^{39818} = 7^2,$$
$$43^{41451} = 2^2 \cdot 5 \cdot 7 \cdot 11^2,$$
$$43^{53596} = 3^3 \cdot 11 \cdot 43,$$
$$43^{12688} = 2^3 \cdot 3 \cdot 7 \cdot 19^2,$$
$$43^{10480} = 2^3 \cdot 3^3 \cdot 5 \cdot 13,$$
$$43^{19831} = 2^8 \cdot 3 \cdot 5 \cdot 11,$$
$$43^{27853} = 2^6 \cdot 3^2 \cdot 5 \cdot 7,$$
$$43^{25154} = 2^5 \cdot 31 \cdot 37,$$
$$43^{9481} = 2^3 \cdot 7 \cdot 11,$$
$$43^{20} = 2^2 \cdot 5^3 \cdot 7^2,$$
$$43^{25418} = 2^5 \cdot 3 \cdot 17 \cdot 19,$$
$$43^{50821} = 5^2 \cdot 41,$$
$$43^{46106} = 2 \cdot 3 \cdot 7 \cdot 11^2,$$
$$43^{14141} = 2 \cdot 3 \cdot 5^2 \cdot 7 \cdot 19,$$
$$43^{26246} = 2 \cdot 3^3 \cdot 5 \cdot 41,$$
$$43^{10795} = 2 \cdot 5^3 \cdot 7 \cdot 11,$$
$$43^{20889} = 5 \cdot 11 \cdot 37,$$

The relation matrix is (cols are relations):

$$\begin{pmatrix}
3 & 0 & 1 & 0 & 1 & 0 & 4 & 1 & 0 & 2 & 0 & 3 & 3 & 8 & 6 & 5 & 3 & 2 & 5 & 0 & 1 & 1 & 1 & 1 & 0 \\
3 & 0 & 0 & 1 & 1 & 0 & 0 & 2 & 0 & 0 & 3 & 1 & 3 & 1 & 2 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 3 & 0 & 0 \\
0 & 4 & 3 & 0 & 0 & 4 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 3 & 0 & 2 & 0 & 2 & 1 & 3 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

The following matrix is made up of rows representing the right kernel:

$$\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 0 & -23 & 3 & 0 & 14 & 18 & 0 & -14 & -13 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & -12 & 1 & 0 & 8 & 10 & 0 & -8 & -8 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & 1 & 2 & -1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 8 & 0 & -15 & 2 & 0 & 8 & 10 & 0 & -8 & -7 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 0 & -25 & 3 & 0 & 14 & 19 & 0 & -15 & -13 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 4 & 0 & -7 & 0 & 0 & 4 & 5 & 0 & -4 & -2 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & -5 & -1 & 0 & 3 & 3 & 0 & -3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 8 & 0 & -16 & 2 & 0 & 9 & 11 & 0 & -9 & -8 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & -5 & 1 & 0 & 2 & 3 & -2 & -2 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 6 & 0 & -15 & 3 & 0 & 8 & 11 & 0 & -8 & -8 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 & 0 & -27 & 3 & 0 & 16 & 20 & 0 & -16 & -13 & 0
\end{pmatrix}$$

The corresponding α_t are:

$$1201200,\ 631400,\ -61600,\ 708400,\ 1232000,\ 323400,\ 277200,\ 754600,\ 169400,\ 662200,\ 1309000.$$

Their gcd is 15400. We check that

$$43^{15400} = 1, \quad 43^{15400/2} = 51174 \neq \pm 1 \pmod{n}.$$

and therefore taking

$$\gcd(51174 - 1, 62389) = 701$$

reveals a non-trivial factor. In fact, $62389 = 701 \cdot 89$.

REFERENCES

[1] Leonard Adleman. "A subexponential algorithm for the discrete logarithm problem with applications to cryptography". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 55–60. DOI: 10.1109/SFCS.1979.2.

[2] Josh Alman and Virginia Vassilevska Williams. "A refined laser method and faster matrix multiplication". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. [Society for Industrial and Applied Mathematics (SIAM)], Philadelphia, PA, 2021, pp. 522–539. DOI: 10.1137/1.9781611976465.32. URL: https://doi-org.colorado.idm.oclc.org/10.1137/1.9781611976465.32.

[3] Don Coppersmith, Andrew M. Odlzyko, and Richard Schroeppel. "Discrete logarithms in GF(p)". In: *Algorithmica* 1.1 (1986), pp. 1–15. ISSN: 0178-4617. DOI: 10.1007/BF01840433. URL: https://doi-org.colorado.idm.oclc.org/10.1007/BF01840433.

[4] Richard Crandall and Carl Pomerance. *Prime numbers*. Second. A computational perspective. Springer, New York, 2005, pp. xvi+597. ISBN: 978-0-387-25282-7; 0-387-25282-7.

[5] Martin Ekerå. "On completely factoring any integer efficiently in a single run of an order-finding algorithm". In: *Quantum Information Processing* 20.20 (2021), p. 205. DOI: 10.1007/s11128-021-03069-1. URL: https://doi.org/10.1007/s11128-021-03069-1.

[6] Felix Fontein and Pawel Wocjan. "On the probability of generating a lattice". In: *J. Symbolic Comput.* 64 (2014), pp. 3–15. ISSN: 0747-7171. DOI: 10.1016/j.jsc.2013.12.002. URL: https://doi-org.colorado.idm.oclc.org/10.1016/j.jsc.2013.12.002.

[7] Steven D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, Cambridge, 2012, pp. xiv+615. ISBN: 978-1-107-01392-6. DOI: 10.1017/CBO9781139012843. URL: https://doi-org.colorado.idm.oclc.org/10.1017/CBO9781139012843.

[8] Daniel M. Gordon. "Discrete logarithms in GF(p) using the number field sieve". In: *SIAM J. Discrete Math.* 6.1 (1993), pp. 124–138. ISSN: 0895-4801. DOI: 10.1137/0406010. URL: https://doi-org.colorado.idm.oclc.org/10.1137/0406010.

[9] James L. Hafner and Kevin S. McCurley. "A rigorous subexponential algorithm for computation of class groups". In: *J. Amer. Math. Soc.* 2.4 (1989), pp. 837–850. ISSN: 0894-0347. DOI: 10.2307/1990896. URL: https://doi-org.colorado.idm.oclc.org/10.2307/1990896.

[10] David Harvey and Joris van der Hoeven. "Integer multiplication in time $O(n \log n)$". In: *Ann. of Math. (2)* 193.2 (2021), pp. 563–617. ISSN: 0003-486X. DOI: 10.4007/annals.2021.193.2.4. URL: https://doi-org.colorado.idm.oclc.org/10.4007/annals.2021.193.2.4.

[11] M. Kraitchik. *Théorie des Nombres, Vol. 1*. Gauthier-Villars, 1922.

[12] A. K. Lenstra. "Fast and rigorous factorization under the generalized Riemann hypothesis". In: *Nederl. Akad. Wetensch. Indag. Math.* 50.4 (1988), pp. 443–454. ISSN: 0019-3577.

[13] H. W. Lenstra Jr. "Factoring integers with elliptic curves". In: *Ann. of Math. (2)* 126.3 (1987), pp. 649–673. ISSN: 0003-486X. DOI: 10.2307/1971363. URL: https://doi-org.colorado.idm.oclc.org/10.2307/1971363.

[14] H. W. Lenstra Jr. and Carl Pomerance. "A rigorous time bound for factoring integers". In: *J. Amer. Math. Soc.* 5.3 (1992), pp. 483–516. ISSN: 0894-0347. DOI: 10.2307/2152702. URL: https://doi-org.colorado.idm.oclc.org/10.2307/2152702.

[15] Gary L. Miller. "Riemann's hypothesis and tests for primality". In: *J. Comput. System Sci.* 13.3 (1976), pp. 300–317. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(76)80043-8. URL: https://doi-org.colorado.idm.oclc.org/10.1016/S0022-0000(76)80043-8.

[16] T. Mulders and A. Storjohann. "Certified dense linear system solving". In: *J. Symbolic Comput.* 37.4 (2004), pp. 485–510. ISSN: 0747-7171. DOI: 10.1016/j.jsc.2003.07.004. URL: https://doi-org.colorado.idm.oclc.org/10.1016/j.jsc.2003.07.004.

[17] J. E. Nymann. "On the probability that k positive integers are relatively prime". In: *J. Number Theory* 4 (1972), pp. 469–473. ISSN: 0022-314X. DOI: 10.1016/0022-314X(72)90038-8. URL: https://doi.org/10.1016/0022-314X(72)90038-8.

[18] A. M. Odlyzko. "Discrete logarithms in finite fields and their cryptographic significance". In: *Advances in cryptology (Paris, 1984)*. Vol. 209. Lecture Notes in Comput. Sci. Springer, Berlin, 1985, pp. 224–314. DOI: 10.1007/3-540-39757-4_20. URL: https://doi-org.colorado.idm.oclc.org/10.1007/3-540-39757-4_20.

[19] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*. https://www.sagemath.org. 2022.

[20] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. A Wiley-Interscience Publication. John Wiley & Sons, Ltd., Chichester, 1986, pp. xii+471. ISBN: 0-471-90854-1.

[21] Martin Seysen. "A probabilistic factorization algorithm with quadratic forms of negative discriminant". In: *Math. Comp.* 48.178 (1987), pp. 757–780. ISSN: 0025-5718. DOI: 10.2307/2007842. URL: https://doi-org.colorado.idm.oclc.org/10.2307/2007842.

[22] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994)*. IEEE Comput. Soc. Press, Los Alamitos, CA, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700. URL: https://doi-org.colorado.idm.oclc.org/10.1109/SFCS.1994.365700.

[23] Arne Storjohann. "Near Optimal Algorithms for Computing Smith Normal Forms of Integer Matrices". In: *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*. ISSAC '96. Zurich, Switzerland: Association for Computing Machinery, 1996, pp. 267–274. ISBN: 0897917960. DOI: 10.1145/236869.237084. URL: https://doi.org/10.1145/236869.237084.

[24] A. E. Western and J. C. P. Miller. *Tables of indices and primitive roots*. Royal Society Mathematical Tables, Vol. 9. Published for the Royal Society at the Cambridge University Press, London, 1968, pp. liv+385.