Research Article

Simran Tinani* and Joachim Rosenthal

A deterministic algorithm for the discrete logarithm problem in a semigroup

https://doi.org/10.1515/jmc-2021-0022
received June 13, 2021; accepted June 08, 2022

Abstract: The discrete logarithm problem (DLP) in a finite group is the basis for many protocols in cryptography. The best general algorithms which solve this problem have a time complexity of $O(\sqrt{N}\log N)$ and a space complexity of $O(\sqrt{N})$, where N is the order of the group. (If N is unknown, a simple modification would achieve a time complexity of $O(\sqrt{N}(\log N)^2)$.) These algorithms require the inversion of some group elements or rely on finding collisions and the existence of inverses, and thus do not adapt to work in the general semigroup setting. For semigroups, probabilistic algorithms with similar time complexity have been proposed. The main result of this article is a deterministic algorithm for solving the DLP in a semigroup. Specifically, let x be an element in a semigroup having finite order N_x. The article provides an algorithm, which, given any element $y \in \langle x\rangle$, provides all natural numbers m with $x^m = y$, and has time complexity $O(\sqrt{N_x}(\log N_x)^2)$ steps. The article also gives an analysis of the success rates of the existing probabilistic algorithms, which were so far only conjectured or stated loosely.

Keywords: discrete logarithm problem, semigroups, complexity of algorithms

MSC 2020: 20M13, 68Q25, 94A60

1 Introduction

Let G be a group and assume $x, y \in G$ are two elements of the group. We refer to x as the base element. The discrete logarithm problem (DLP) asks for the computation of an integer $m \in \mathbb{Z}$ (assuming such integers exist) such that $x^m = y$. The DLP plays an important role in a multitude of algebraic and number theoretic cryptographic systems. Its use was introduced in the Diffie–Hellman protocol for public key exchange [1] and has since seen a tremendous amount of development, generalizations, and extensions [2]. Many modern-day systems for public key exchange use the DLP in a suitable group. The most commonly used groups have been the multiplicative group of finite fields and the group of points on an elliptic curve. The DLP in Jacobians of hyperelliptic curves and more general abelian varieties has also been studied extensively [3].

In this article, we compute complexities using group multiplications as one fundamental step. Thus, an exponentiation x^e is performed in $O(\log e)$ steps. We will use the fact that for two lists of length n in which a match exists, a match can be found in $O(n\log n)$ steps using standard sorting and searching algorithms (for details, the interested reader may refer to ref. [4]). For a general finite group of order N, there exist algorithms that solve the DLP in $O(\sqrt{N}\log N)$ steps. Such algorithms are said to produce a square root attack. The most well-known examples are Shank's baby step-giant step algorithm [5] and the Pollard-Rho

* **Corresponding author: Simran Tinani,** Institute of Mathematics, University of Zurich, Zürich, Switzerland, e-mail: simran.tinani@math.uzh.ch
Joachim Rosenthal: Institute of Mathematics, University of Zurich, Zürich, Switzerland, e-mail: rosenthal@math.uzh.ch

algorithm [6]. Note that Shank's algorithm is a deterministic algorithm having time complexity $O(\sqrt{N}\log N)$ and space complexity $O(\sqrt{N})$. In contrast, Pollard's algorithm is a probabilistic algorithm having time complexity $O(\sqrt{N}\log N)$ group multiplications and space complexity $O(1)$. If N is unknown, a simple modification of these algorithms would achieve a time complexity of $O(\sqrt{N}(\log N)^2)$.

Elliptic curve groups have been widely implemented in practice since for a carefully selected elliptic curve group, the best known classical algorithm for solving DLP has running time $O(\sqrt{N}\log N)$, where N is the group order. This is in contrast to many other finite groups such as the multiplicative group of a finite field and the group of invertible matrices over a finite field where algorithms with subexponential running time are known [7].

In cryptography the Diffie–Hellman protocol using a finite group has been generalized to situations where the underlying problem is a DLP in a semigroup or even to situations where a semigroup acts on a set [8,9]. The interested reader will find more material in a recent survey by Goel et al. [10].

It is naturally interesting to ask whether the DLP also has a square root attack in more generalized structures such as semigroups. Here, we define a semigroup as any set of elements with an associative binary operation. Since the best algorithms for the DLP all make use of the existence of inverses, it is unclear whether they can be generalized to a semigroup. However, when a special type of semigroup element, called a torsion element, is used as the base, it turns out that the DLP is reducible in polynomial time to the DLP in a finite group. A torsion element is one whose powers eventually repeat to form a cycle, and will be defined more precisely in Section 2. This section also elaborates more on why the standard collision-based algorithms are not directly adaptable to the semigroup case. A semigroup in which every element is torsion is called a torsion semigroup.

The DLP in semigroups with a torsion base element, in a classical setting, was first discussed by Monico [11] in 2002, and later in a article by Banin and Tsaban [12] in 2016. While the discussion in the present article is entirely on classical algorithms, it is also worth mentioning the paper [13], where the authors independently provide a quantum algorithm that solves the DLP in a torsion semigroup.

Both the algorithm of Monico and the one of Banin and Tsaban are probabilistic and might fail with low probability. Furthermore, some of their methods are heuristic, dependent on an oracle or some additional assumption, and their success rates and expected number of steps are either conjectured or stated loosely. It is therefore of interest to come up with an algorithm which deterministically computes the discrete logarithm in a semigroup. In this regard, we like to make some analogy to the problem of determining if an integer is a prime number, a problem of great importance in cryptography. Nowadays, in practice the algorithm of Miller [14] and Rabin [15] has been implemented for many years. Still it was a great result when Agrawal et al. [16] came up with a deterministic polynomial time algorithm to achieve this goal.

A key step in finding the discrete logarithm in a semigroup is computing the cycle length of an element. Both the algorithms of refs [12] and [11] rely on computing some multiple of the cycle length, and then removing "extra" factors by taking greatest common divisors (gcd's) until the cycle length is obtained. Once the cycle length value is obtained, the discrete logarithm may easily be computed with a few more simple steps. While Monico does not provide further elaboration on how this is done, the paper by Banin and Tsaban bridges this knowledge gap by showing how the problem is reduced to a DLP in a group once the cycle length and start values are known. Denote by N_x the order of x (formally defined in Definition 4). The complexity of the algorithm in ref. [12] is $O(\sqrt{N_x}(\log N_x)^2\log\log N_x)$, and that of the one in ref. [11] is $O(\sqrt{N_x}(\log N_x)^2)$. While both of the existing methods seem to succeed with high probability for practical values, we show that the process of taking successive gcd's/factors is unnecessary, and that one can deterministically find the cycle length. The main contribution of this article will be a deterministic algorithm for computing the discrete logarithm of an element y in some semigroup S with respect to some torsion base element $x \in S$. The time complexity of our algorithm is $O(\sqrt{N_x}(\log N_x)^2)$.

The article is structured as follows: After providing preliminaries and basic definitions in Section 2, we will analyse in Section 3 the success rates and expected number of steps involved in the probabilistic algorithms for cycle length by Banin and Tsaban (Algorithm 1) and Monico (Algorithm 3.1).

In Section 4, which is the main section of this article, we provide a deterministic algorithm to calculate the cycle length L_x of a torsion element x of a semigroup and thus to also solve the DLP, without the use of an oracle. This algorithm has complexity $O(\sqrt{N_x}\cdot(\log N_x)^2)$. For completeness, we will also demonstrate the use of Pohlig–Hellman algorithm [17] for a semigroup.

2 Preliminaries

A semigroup S is a set together with an associative binary operation. Like in group theory where a torsion group consists of elements of finite order only we define:

Definition 1. (Torsion element). Let S be a semigroup. An element $x \in S$ is called a torsion element if the sub-semigroup $\langle x\rangle \coloneqq \{x^k | k \in \mathbb{N}\}$ generated by x is finite. S is called a torsion semigroup if every $x \in S$ is a torsion element.

Throughout the article the following definitions will be assumed:

Definition 2. (Cycle start). Let $x \in S$. The cycle start s_x of x is defined as the smallest positive integer such that $x^{s_x} = x^b$ for some $b \in \mathbb{N}$, $b > s_x$.

Definition 3. (Cycle length). Let $x \in S$. The cycle length L_x of x is defined as the smallest positive integer such that $x^{s_x+L_x} = x^{s_x}$.

Definition 4. (Element order) Let $x \in S$. With notation as above, we define the order N_x of x as the cardinality of the sub-semigroup $\langle x\rangle$. Note that $N_x = s_x + L_x - 1$.

Definition 5. (Semigroup DLP). Let S be a semigroup and $x \in S$. The semigroup DLP is defined as follows. Given $y \in \langle x\rangle \coloneqq \{x^k | k \in \mathbb{N}\}$, find all $m \in N$ such that $x^m = y$.

We state below a key result first proved in ref. [12].

Lemma 1. [12] *Let S be a semigroup and $x \in S$ be an element with cycle start s_x. The set of powers $G_x = \{x^{s_x+k} | k \geq 0\}$ of x forms a finite cyclic group. The identity element of G_x is given by x^{tL_x}, where t is the minimum positive integer such that $x^{tL_x} \in G_x$.*

The following result is stated in ref. [11] in a slightly different formulation. We provide an equivalent proof based on the group structure of G_x.

Lemma 2. [11] *Let $x \in S$ have cycle start s_x and cycle length L_x. For all integers $n, m \geq s_x$, we have $x^m = x^n \Leftrightarrow n \equiv m \bmod L_x$.*

Proof. We can assume without loss of generality that $n \geq m$, and so we can write $n = m + kL_x + u$, with $k \geq 0$ and $0 \leq u < L_x$. First suppose that $n \equiv m \bmod L_x$, i.e. $u = 0$. Since $m, n \geq s_x$, we have $x^n = x^{m+kL_x} = x^m$.

Conversely, if $x^n = x^m$, write $n_1 = n - s_x \geq 0$, and $m_1 = m - s_x \geq 0$. We have

$$x^{s_x+m_1} = x^{s_x+n_1} = x^{s_x+m_1+kL_x+u} = x^{s_x+m_1+u}.$$

Now, without loss of generality, $m_1 \geq s_x$, because if not, one can always increment m_1 and n_1 by multiples of L_x until this happens. So, we can assume that x^{m_1} lies in G_x and is thus invertible. We multiply by the inverse on both sides to finally obtain

$$x^{s_x} = x^{s_x+u}.$$

Thus, we must have $u = 0$ or $n \equiv m \bmod L_x$, as required. □

Remark 1. It becomes clear from the above discussion that the standard collision-based algorithms for order and discrete log computations in a group do not adapt directly to a general semigroup. Collision-based algorithms for the computation of the order N of a group element x (for instance, see ref. [18]) are based on the principle that whenever N can be expressed in the form $N = A - B$ for non-negative integers A and B, the collision $x^A = x^B$ always occurs. However, this principle does not work in a semigroup, where there are two independent components of the order. More specifically, for a semigroup element x with cycle length L_x and cycle start s_x, whenever L_x may be expressed in the form $A - B$ for non-negative integers A and B, the equality $x^A = x^B$ holds if and only if $A, B \geq s_x$. As an example, consider a semigroup element x with cycle length $L_x = 12$ and cycle start $s_x = 5$. Then, $L_x = 15 - 3$, but $x^{15} \neq x^3$. Thus without prior knowledge of the cycle start, the semigroup order N_x or cycle length L_x cannot directly be found using the same collision-based algorithms for groups.

Similarly, collision-based algorithms fail for discrete log computations in a semigroup. As an example, consider a semigroup element x with cycle length $L_x = 15$ and cycle start $s_x = 10$, and suppose that the discrete log of $y = x^5$ is to be found. Then $y \cdot x^6 = x^{11} = x^{26}$ is obtained as a collision. However, unlike in the group case, the conclusion $y = x^{26-6} = x^{20}$ is wrong since $x^5 \neq x^{20}$. This happens because even though x is torsion and forms a cycle of powers, it is not invertible.

This concludes the prerequisite knowledge on torsion elements in semigroups. In the next section, we study the existing probabilistic algorithms for cycle lengths and analyse their assumptions, working, and complexities.

3 Existing probabilistic algorithms

3.1 Banin and Tsaban's algorithm

In this section, we study the probabilistic algorithm described in ref. [12] for computing the cycle length of a torsion element in a semigroup. While the authors of the original article describe their theory only for torsion semigroups, it will become clear that the same discussion holds true for any semigroup when the base element chosen is torsion.

Let S be a semigroup and x be a torsion element of S. Let s_x denote the cycle start of x and L_x its cycle length. Then, recall from Lemma 1 that $G_x := \{x^{s_x}, x^{s_x+1}, \ldots, x^{s_x+L_x-1}\}$ is a cyclic group, and that it has order L_x. The authors of ref. [12] assume the availability of a "Discrete Logarithm Oracle" for the group G_x, which returns values $\log_x h$ for $h \in G_x$. They state that these values need not be smaller than the group order but are polynomial in the size of G_x and the element x. The representation of the identity in G_x is unknown, and a method to compute inverses is not available.

The authors claim that the well-known algorithms for discrete logarithm computations in groups do not explicitly require inverses, or can easily be modified to work without the use of inverses. While it is true that these algorithms make use of mainly the existence of inverses rather than their explicit computation, we believe that the fact that easy modification is possible is not immediate without some justification. In fact, it will become clear in the later sections that the modified baby-step-giant-step algorithm devised by Monico [11] (and also the deterministic algorithm presented in Section 4) is a crucial and non-trivial part of any such modification.

We make the following observation from the proof of Lemma 1 found in ref. [12]. For any $k \geq 0$, denote by v_k the smallest positive integer such that

$$v_k L_x \geq 2s_x + k.$$

We then have $x^{v_k L_x - s_x - k} \in G_x$ and

$$x^{s_x+k} x^{v_k L_x - s_x - k} = x^{v_k L_x} = x^{t L_x}, \tag{1}$$

so the inverse of the element x^{s_x+k} of G_x is given by $x^{v_k(L_x)-s_x-k}$. In particular, the computation of inverses requires prior knowledge of the cycle start. As will be explained below, the cycle start may be computed only once the value of the cycle length is known, using a binary search. This explains why the authors insist that their discrete logarithm oracle does not need to use the computation of inverses.

Below, we describe Algorithm 1, which is the algorithm suggested in ref. [12] to compute the order of the group G_x, i.e. the cycle length L_x of x.

Algorithm 1: Banin–Tsaban algorithm for cycle length

Input A semigroup S and a torsion element $x \in S$; a DLP oracle for groups
Output The cycle length L_x of x

1: Initialize $i, j, g, L_x \leftarrow 1$, $N \gg s_x + L_x$. Fix bounds $r > 1$, $s > 1$.
2: **while** $j < s$
  1. Fix a random $z \in \{\lfloor M/2 \rfloor, \ldots, M\}$ and set $h = x^z$.
  2. **while** $i < r$
    (a) Choose a random number $k_i > 0$.
    (b) Use the DLP oracle to compute $k_i' = \log_h(h^{k_i})$.
    (c) Set $g \leftarrow \gcd_{j \le i}(k_j - k_j') = \gcd(\gcd_{j<i}(k_j - k_j'), \quad k_i - k_i')$.
    (d) Set $i \leftarrow i + 1$.
  3. **end while**
  4. Set $L_x \leftarrow \mathrm{lcm}(L_x, g)$, $j \leftarrow j + 1$.
6: Return L_x.

We first note that the authors state complexities in terms of L_x, which are valid when the bound N for N_x is known. If the algorithm fails for a value of N, the authors suggest to double N and try again. In this case, which we will assume from now on, we assert that the complexities need to be taken in terms of N_x instead of L_x. The oracle may be assumed to have the standard complexity of $O(\sqrt{N_x}\log N_x)$ steps for discrete logarithm calculations. Step (2.2(c)) takes $O(\log(\max_{j \le i}(k_j - k_i)) = O(\log N_x)$ integer operations by the assumption on the oracle, which does not contribute to the total complexity. Thus, the total complexity of Step (2.2) comes from the oracle alone and is $O(\sqrt{N_x}\log N_x)$. Now, the authors of ref. [12] remark that r and s can be taken to satisfy $r = O(1)$ and $s = O(\log\log N_x)$. Thus, the total complexity is $O(\log N_x)$ times the complexity of Algorithm 1, and thus $O(\log\log(N_x)\log N_x)$ times the complexity of Step (2.2). Therefore, we obtain the total complexity of $O(\log\log N_x (\log N_x)^2 \sqrt{N_x})$.

Finally, in Algorithm 2, we present the application of the binary search method to find the cycle start once L_x is known. This algorithm is formulated as below for this purpose in ref. [12], though the idea to use a binary search is also originally mentioned in ref. [11].

Algorithm 2: Calculating cycle start (binary search)

Input A semigroup element x with cycle length L_x
Output Cycle start s_x of x

1: Initialize $s_x \leftarrow 1$
2: **while** $x^{s_x+L_x} \neq x^{s_x}$ **do**
  $s_x \leftarrow 2s_x$

3: **end while**
4: Set $a \leftarrow s_x/2$
5: **while** $|a - s_x| \geq 2$
$c \leftarrow (a + s_x)/2$
if $x^{c+L_x} \neq x^c$ **then**
$a \leftarrow c$
else
$s_x \leftarrow c$
6: **end while**

Lemma 3. *Let N_x be the order of the element x. Then Algorithm 2 requires*

$$O((\log N_x)^2)$$

steps.

Proof. Each of Steps (2) and (5) of Algorithm 2 involves $O(\log N_x)$ iterations. Each iteration, in turn, requires $O(\log N_x)$ semigroup multiplications and one comparison. The total complexity is thus $O((\log N_x)^2)$. □

3.2 Monico's algorithm

In his PhD thesis [11], Chris Monico provides a probabilistic algorithm (described below as Algorithm 3.1) that calculates the cycle length of an element in a finite ring of order N. This algorithm makes use of the multiplicative semigroup structure of the finite ring and of the availability of the explicit bound N for every cycle length, and is in fact applicable to any semigroup where such a bound N is available. In this subsection, we analyse this algorithm, provide a more concrete bound on its success rate, and compute its complexity in terms of N. We will discuss this algorithm in terms of torsion semigroups, as opposed to finite rings.

We first note that if $L_x > m$ and the table in Step (2) has repeated entries $x^{q+i_1 m} = x^{q+i_2 m}$, then numbers b_1 and b_2 may not exist below m. In this case, the algorithm needs to be modified to take $g \leftarrow (i_1 - i_2)m$. However, whenever this case does not arise, it can be shown that Steps (3) and (4) are always successful in finding a collision.

We further remark that in Step (6), the list of divisors of g is kept fixed, while g is updated to g/d whenever the condition is satisfied. In the subsequent steps, non-divisors of g/d can be immediately discarded. However, the end result depends on the order in which divisors are tested, which the algorithm does not mention explicitly. However, we note that it is, in fact, possible to restrict the testing to only the prime power divisors of g below B, and with this setting, the optimal performance is obtained by taking divisors in decreasing order. We will assume this set-up for the rest of the analysis. For completeness, we restate Algorithm 3.1 with the above clarifications in Algorithm 3.2.

Algorithm 3.1: Monico's baby step-giant step for cycle length

Input A finite semigroup S with $|S| = N$ and an element $x \in S$
Output The cycle length L_x of x

1: Set $m = \lceil \sqrt{N} \rceil$. Choose a prime $q > N$.
2: For $0 \leq i \leq m$, compute and store in a table the pairs $(i; x^{q+im})$.
Sort the table by the second component.
3: Find the least positive integer b_1 such that x^{q+b_1} is in the table: $x^{q+b_1} = x^{q+a_1 m}$. (Note: $0 < b_1 < m$.)

4: Find the least positive integer b_2 such that x^{2q+b_2} is in the table: $x^{2q+b_2} = x^{q+a_2m}$. (Again, $0 < b_2 < m$.)
5: Compute $g = \gcd(a_1m - b_1, a_2m - b_2 - q)$.
6: For each divisor d of g below some bound B, do the following.
If $x^{N+g/d} = x^N$:
set $g \leftarrow g/d$;
7: Output $L_x = g$ and stop.

Algorithm 3.2: Restated: Monico's baby step-giant step for cycle length

Input A finite semigroup S with $|S| = N$ and an element $x \in S$
Output The cycle length L_x of x
1: Set $m = \lceil \sqrt{N} \rceil$. Choose a prime $q > N$.
2: For $0 \le i \le m$, compute and store in a table the pairs $(i; x^{q+im})$.
Sort the table by the second component. If a collision $x^{q+i_1m} = x^{q+i_2m}$ occurs, set $g = (i_1 - i_2)m$ and go to Step (6).
3: Find the least positive integer b_1 such that x^{q+b_1} is in the table: $x^{q+b_1} = x^{q+a_1m}$. (Note: $0 < b_1 < m$.)
4: Find the least positive integer b_2 such that x^{2q+b_2} is in the table: $x^{2q+b_2} = x^{q+a_2m}$. (Again, $0 < b_2 < m$.)
5: Compute $g = \gcd(a_1m - b_1, a_2m - b_2 - q)$.
6: Fix a bound B and compute all the divisors of g below B. Denote these by $d_1 > d_2 > \ldots > d_r$.
7: For $i = 1, \ldots, r$, do the following.
If $x^{N+g/d_i} = x^N$:
set $g \leftarrow g/d_i$.
8: Output $L_x = g$ and stop.

Note that Step (2) involves $O(\log N)$ steps to compute x^q and x^m and another $O(\sqrt{N})$ multiplications to compute $x^q, x^q \cdot x^m, x^q \cdot x^{2m}, \ldots, x^q \cdot x^{m^2}$. Step (3) involves at most m multiplications $x^{q+1} = x^q \cdot x, x^{q+1} \cdot x, \ldots, x^{q+m-1}$, with complexity $O(\sqrt{N})$, and match-finding with the first list, with complexity $O(\sqrt{N}\log N)$ with standard sorting and search algorithms. The same is true for Step (4). Step (5) has complexity $O(\log\max(a_1m - b_1, a_2m - b_2 - q)) = O(\log N)$ and so does not contribute to the overall complexity. Step (6) involves B iterations of a multiplication and an exponentiation $x^{g/d}$, and thus has a time complexity of $O(B(\log g + 1)) = O(B \log N)$ multiplications.

In the original work, Monico states that the bound B of Algorithm 3.1 can always be chosen so that $B < \sqrt{a_1m - b_1}$. We remark that this claim does not hold in the current setting of the algorithm. For example, with a cycle length value of 4, and $a_1m - b_1 = 104$, $a_2m - b_2 - q = 52$, we obtain $g = 52$. If $B < \sqrt{a_1m - b_1} = \sqrt{104} < 11$, then we would only test divisors d below 11, and would never factor out 13 to obtain the true cycle length. For such a bound to work, one needs to modify the algorithm to test both divisors d and g/d in Step (6). However, we will show in Lemma 4 that it is almost always sufficient to take B to be a reasonably large fixed constant, thus the complexity of Step (6) can be counted as $O(\log N)$, and does not contribute to the overall complexity. Thus, the overall time complexity is $O(\sqrt{N}\log N)$. If N is unavailable, the algorithm can also be modified to update the value of N step-by-step until a large enough value is found. In this case, Algorithm 3.1 has a total complexity of $O(\sqrt{N_x}(\log N_x)^2)$.

Furthermore, Monico suggests a modification to the above algorithm, viz. to find several such a_i and b_i and compute all the gcd's. It is clear that this suggestion is exactly the method used in Banin and Tsaban's algorithm as discussed in Section 3.1.

We now analyse the probability of success. The algorithm first looks for collisions of the form $x^{q+a_1m} = x^{q+b_1}$. The working principle is that in this case, the cycle length L_x divides $a_1m - b_1$. Similarly, if also $x^{q+a_2m} = x^{2q+b_2}$ then $g = \gcd(a_1m - b_1, a_2m - b_2 - q)$ is a multiple of L_x.

So far, the process is essentially the same in both Algorithms 1 and 3.1: while the former uses a discrete logarithm oracle to obtain multiples of the cycle length, the latter directly finds these multiples by finding collisions. However, in Algorithm 3.1, we do not proceed with computing multiple factors of L_x, but work with the fixed multiple g of L_x, whereas in Algorithm 1 this multiple shrinks several times.

Algorithm 3.1 then proceeds by fixing a bound B and iterating over every number d below B to check if $d|g$. If yes, it executes the next part, i.e. checks if $x^{N+D/d} = x^N$, and if this holds, it sets $D \leftarrow D/d$. Note that if the factorization of the number g is known (or if g can be factored in time negligible compared to $O(\sqrt{N})$, then we do not need this fixed bound B, and can instead iterate over every prime factor d of g. It is well-known that the number of prime factors of g counted with multiplicity is $O(\log g)$, so Step (5) of the algorithm can find L_x in $O(\log N)$ steps. However, in general, factoring g may be difficult, so we assume from here on that the algorithm proceeds by fixing a bound B for the divisors of g. Below we analyse the probability of the algorithm succeeding in terms of B and g.

Lemma 4. *The probability that Algorithm 3.2 succeeds is bounded below by* $\left(1 - \frac{1}{B}\right)^{\log g}$.

Proof. We write $g = L_x \cdot F$ for some number F and suppose that the algorithm fails. This means that there is a divisor, and hence also a prime power divisor of F, which the algorithm fails to factor out. Let p be a prime dividing F, α_p denote its largest power dividing F, and β_p be its largest power below the fixed bound B. So, we have $p^{\alpha_p}|F$, $p^{\alpha_p+1}\nmid F$, $p^{\beta_p} < B$, $p^{\beta_p+1} > B$.

Note that the number of times the algorithm divides g by p is

$$\sum_{i=1}^{\beta_p} i = \beta_p \cdot (\beta_p + 1)/2.$$

Since divisors are taken in decreasing order, we must have $\beta_p \cdot (\beta_p + 1)/2 < \alpha_p$ if the algorithm fails. So, the algorithm succeeds as long as $\beta_p \cdot (\beta_p + 1)/2 \geq \alpha_p$ for every prime divisor p of F. Thus, the probability of success for the algorithm can be bounded below by

$$\prod_{p|g} \text{Prob}\left(\frac{\beta_p \cdot (\beta_p + 1)}{2} \geq \alpha_p\right).$$

Write $v_p = \frac{\beta_p(\beta_p+1)}{2}$ for simplicity. We may assume that g is a random multiple of L_x below the bound B, so F is a random number in $\left\{1, \ldots, \frac{B}{L_x}\right\}$. We have,

$$\begin{aligned}\text{Prob}(\alpha_p \leq v_p) &= 1 - \text{Prob}\left(p^{v_p+1}|F\right) \\ &= 1 - \left(\frac{B/L_x}{p^{v_p+1}(B/L_x)}\right) \\ &= 1 - 1/p^{v_p+1} = 1 - \frac{1}{p^{\frac{\beta_p(\beta_p+1)}{2}+1}}.\end{aligned}$$

Hence, a lower bound for the probability of the algorithm's success is

$$\prod_{p|F}\left(1 - \frac{1}{p^{\frac{\beta_p\cdot(\beta_p+1)}{2}+1}}\right).$$

Now, we have

$$p^{\beta_p+1} > B \quad \Leftrightarrow \quad \frac{1}{p^{\beta_p+1}} < \frac{1}{B} \quad \Rightarrow \quad 1 - \frac{1}{p^{\frac{\beta_p(\beta_p+1)}{2}+1}} > 1 - \frac{1}{B^{\frac{\beta_p}{2}+1}} > 1 - \frac{1}{B}.$$

We further make the following observation. Let $\omega(n)$ denote the number of distinct prime divisors of integer n (note, however, that the same statement also holds if counted with multiplicity). Then clearly, $2^{\omega(n)} \leq n$, and so, taking logarithms, $\omega(n) \leq \log_2 n$.

Collecting all the above results, we conclude that the probability of success Prob (success) of Algorithm 3.1 is bounded below as follows:

$$\begin{aligned}\text{Prob (success)} &\geq \prod_{p|F}\left(1-\frac{1}{B}\right)\\ &= \left(1-\frac{1}{B}\right)^{\omega(F)} \geq \left(1-\frac{1}{B}\right)^{\log F}\\ &\geq \left(1-\frac{1}{B}\right)^{\log g}.\end{aligned}$$

□

Note that this bound shows that Algorithm 3.1 is indeed successful with overwhelming probability, as conjectured by the author. For example, with $B = 10^6$, even when g is several orders of magnitude larger than B, say $g = 2^{4,000}$, the probability of success is greater than 99.6%, by the bound derived in Lemma 4.

4 Deterministic solution of the DLP

The solution of the DLP in a semigroup involves two parts: the calculation of the cycle length and start of the base element x, and the use of this value to find the discrete log.

4.1 Deterministic algorithm for cycle length computation

We now present our deterministic algorithm for the computation of the cycle length. It works by finding a suitable collision, and also guarantees finding the actual cycle length rather than just a multiple of it, in a fixed number of steps.

Algorithm 4: Deterministic algorithm for cycle length

Input A semigroup S and a torsion element $x \in S$. Assume N_x is the order of x.
Output Cycle length L_x of x

1: Initialize $N \leftarrow 1$.
2: Set $q \leftarrow \lceil \sqrt{N} \rceil$.
3: Compute, one by one, $x^N, x^{N+1}, \ldots, x^{N+q}$ and check for the equality $x^N = x^{N+j}$ at each step $j \geq 1$. Store these values in a table as pairs $(N+j, x^{N+j})$, $0 \leq j < q$. If $x^N = x^{N+j}$ for any $j < q$, then set $L_x \leftarrow j$ and end the process. If not, proceed to the next step.
4: For $0 \leq i \leq q$, compute, one by one, the values $x^{N+q}, x^{N+2q}, \ldots, x^{N+iq}$ and at each step i, look for a match in the table of values calculated in Step (3).
5: Suppose that a match $x^{N+iq} = x^{N+j}$ is found, and i is the smallest integer such that this happens. Set $L_x \leftarrow iq - j$ and end the process.
6: If no match is found in Steps (3) or (5), set $N \leftarrow 2 \cdot N$ and go back to Step (2).

Theorem 1. *Let S be a semigroup and $x \in S$ a torsion element with order N_x. If an upper bound on N_x is known, Algorithm 4 returns the correct value of the cycle length L_x with*

$$O(\sqrt{N_x} \cdot (\log N_x)^2)$$

steps. The total space complexity is $O(\sqrt{N_x})$ semigroup elements.

Proof. We first assume $N \geq \max(L_x, s_x)$ and show that Steps (1)–(5) succeed in finding L_x. We have $q = \lceil \sqrt{N} \rceil$. If $L_x < q$, then the equality $x^N = x^{N+L_x}$ is found in the first step and the statement of the theorem follows. Else if $L_x \geq q$, we can write uniquely

$$L_x = iq - j,$$

for some positive integers $i > 0$, $0 \leq j < q$. Now, we must have $i \leq q$, because otherwise if $i \geq q + 1$, we would have

$$L_x \geq (q + 1)q - j > q^2 + q - q = q^2 \geq N,$$

a contradiction.

We have

$$\begin{aligned} & L_x = iq - j, \ 0 < i \leq q, \ 0 \leq j < q \\ \Rightarrow \ & N + j + L_x = N + iq \\ \Rightarrow \ & x^{N+j} = x^{N+j+L_x} = x^{N+iq}, \end{aligned}$$

where the last step follows because $N > s_x$ by assumption. So, such a collision always occurs between elements of the two lists in the algorithm.

We now claim that for the smallest such integer i computed in Step (5) of Algorithm 4, $L_x = iq - j$.

To see this, let i be the smallest positive integer such that

$$x^{N+j} = x^{N+iq}.$$

Also let $L_x = i'q - j'$, $0 < i' \leq q$, $0 \leq j' < q$. We have already shown above that such integers i' and j' exist for our choice of N. By the definition of L_x, we must have $L_x | iq - j$. Now suppose that $i' > i$. Then,

$$i'q - j' \geq (i + 1)q - j' = iq + (q - j') > iq \geq iq - j.$$

But, $L_x = i'q - j' | iq - j$, so we must have $iq - j = i'q - j'$. Since $i' > i$, this means that

$$q \leq (i' - i)q = (j' - j) < j',$$

which is a contradiction because $0 \leq j' < q$. So, we must have $i' = i$, $j' = j$. This proves the claim.

We have shown above that the algorithm finds the correct cycle length when $N > \max(s_x, L_x)$. Since the algorithm doubles the value of N until a match is found, it always terminates and outputs the correct cycle length. We now look at the time complexity.

For a given N, Step (2) involves one exponentiation, or $O(\log N)$ multiplications to find x^N and then at most another $q = O(\sqrt{N})$ multiplications and equality checks for $x^N \cdot x, x^N \cdot x^2, \ldots, x^N \cdot x^q$. This step also needs a storage space of at most $q = O(\sqrt{N})$ elements. Step 5 needs one exponentiation or $O(\log N)$ multiplications to find x^q, and then another $q = O(\sqrt{N})$ multiplications to find $x^{N+q} \cdot x^q, x^{N+q} \cdot x^{2q} \ldots, x^{N+q^2}$. Finding matches in Steps (3) and (5) can be done in $O(q \log q) = O(\sqrt{N} \log \sqrt{N})$ comparisons with the use of sorting and efficient look-up methods. Thus, clearly, Steps (1)–(5) in Algorithm 4 have a total complexity of $O(\sqrt{N} \log N)$.

Moreover, the algorithm starts at $N = 1$ and doubles N until the cycle length is found, i.e. until $N > \max(s_x, L_x)$. Thus, the number of times Steps (2)–(5) are performed is

$$\lceil \log(\max(L_x, s_x)) \rceil = O(\max(\log(L_x), \log(s_x))) = O(\log N_x)$$

Thus, the total number of steps involved is

$$O(\sqrt{N_x} \cdot (\log N_x)^2).$$

Clearly, Step (3) involves the storage of $q = \lceil \sqrt{N} \rceil = O(\sqrt{\max(s_x, L_x)}) = O(\sqrt{N_x})$ elements, so this value gives the total space complexity. This completes the proof. □

Remark 2. If a bound N on the order N_x is known *a priori*, then Algorithm 4 can clearly be completed in a single round, with time complexity $O(\sqrt{N}\cdot(\log N))$.

Remark 3. For the case of a group, there exist better algorithms for the computation of the order of an element even when the total group order is unbounded. For instance, Algorithm 3.3 in ref. [18] uses a growth function $d(t)$, which generalizes the square root function used above, to compute the order N of a group element x, and achieves time and space complexities of $O(\sqrt{N})$, thus eliminating the additional $\log N$ multiplier introduced by the method in Algorithm 4.

However, this method fails when used for a general semigroup due to the presence of two independent unknown components of the order. To see this, note that the algorithm would need to be modified for a semigroup as follows. At stage t, one has $g(t-1) \le N_x < g(t)$. On the completion of the baby steps, one has a table with the powers $x^{g(t)}, x^{g(t)+1}, \ldots, x^{g(t)+b(t)}$ (the addition of $g(t)$ is necessary in the semigroup case to ensure that the loop is entered). The giant steps compute $x^{g(t)+g(t-1)+\cdot b(t)}, x^{g(t)+g(t-1)+2\cdot b(t)}, \ldots, x^{g(t)+g(t-1)+d(t)\cdot b(t)} = x^{2g(t)}$. Now, while N_x is guaranteed to have a unique expression as $g(t-1) + ib(t) - j$ with $0 < i \le d(t)$ and $0 \le j \le b(t)$, this does not necessarily lead to a collision. In fact, if $b(t) < L_x < g(t-1)$ and $2L_x > g(t) = g(t-1) + d(t)\cdot b(t)$, then neither the baby steps nor the giant steps leads to a collision, and the cycle length is never found (note that this can happen only if $L_x > s_x$). Moreover, if a collision $x^{g(t)+g(t-1)+ib(t)} = x^{g(t)+j}$ is obtained in the giant step phase, the only conclusion that can be drawn is that $L_x | g(t-1) + ib(t) - j$. If instead we forced the condition $g(t-1) \le N_x < g(t)$, a collision again may never occur because there is no control on the cycle start (For instance, in matrix semigroups over finite simple semirings, the cycle start is often found to be much larger than the cycle length. In such cases, adapting group-based algorithms would fail.) See Remark 1 for further details.

4.1.1 Experimental results for cycle length computations

We used Algorithm 4 to compute cycle length values in several common semigroups, such as matrix semigroups over finite fields, matrix semigroups over the finite simple semiring S_{20} (see ref. [19] for a construction and ref. [9] for the addition and multiplication tables), and the symmetric and alternating groups (where the cycle length is precisely the order of the element). We further used the obtained cycle lengths to compute the cycle start values using Algorithm 2. The working code may be found at https://github.com/simran-tinani/semigroup-cycle-length.

4.2 Solving the DLP once the cycle length is known

In this section, we demonstrate the solution of the DLP for a torsion element x in the semigroup S once the cycle length is known. As before let N_x be the order of the sub-semigroup $\langle x \rangle$, let L_x be the cycle length of the torsion element x (which we assume is already computed), and let $y \in \langle x \rangle$ be an element.

In ref. [12], the authors demonstrate the next steps in solving for $\log_x(y)$, via a reduction to a DLP in the group G_x, once L_x and s_x are known. The procedure is described in Algorithm 5, which has been adapted from the original formulation in ref. [12].

Algorithm 5: Algorithm for discrete logarithm

Input A semigroup S, a torsion element $x \in S$, with cycle length L_x and cycle start s_x, and $y \in S$ with $y = x^m$

Output The discrete logarithm m of y with base x

1: Compute $t = \left\lceil \frac{s_x}{L_x} \right\rceil$ and define $x' = x^{tL_x+1} \in G_x$.
2: Find the minimum number $0 \le b \le t$ such that $y' = y \cdot x^{bL_x} \in G_x$ using binary search.
3: Use Shank's baby step–giant step algorithm for the group $\langle x' \rangle \subseteq G_x$ to compute $m' \in \{0, 1, \ldots, L_x - 1\}$ such that $(x')^{m'} = y'$.
4: Find the maximum number $c \ge 0$ such that $x^{(tL_x+1)m'-cL_x} \in G_x$ using binary search.
5: Return $m = m'(tL_x + 1) - (b + c)L_x$.

Since authors of ref. [12] do not give an explicit proof of correctness Step (5) in Algorithm 5, we provide it in Theorem 2. Before this, we will need the following technical result.

Lemma 5. *Let L_x be the cycle length of $x \in S$, and n, a, and a' be fixed positive integers. Suppose that $x^{bL_x+n} = x^a \in G_x$, where b is the minimum number such that $x^{bL_x+n} \in G_x$, and $x^{n-cL_x} = x^{a'} \in G_x$, where c the maximum number such that $x^{n-cL_x} \in G_x$. Then*

$$bL_x + n \le a \quad \text{and} \quad n - cL_x \le a'.$$

Proof. First let $x^{bL_x+n} = x^a$ with b minimal such that $x^{bL_x+n} \in G_x$. Suppose, to the contrary, that $bL_x + n > a$. We must have, by the minimality of b, $x^{(b-1)L_x+n} \notin G_x$, so $(b-1)L_x + n < a$.

$$\begin{aligned} &\text{But, } x^{bL_x+n} = x^a \in G_x \\ &\Rightarrow\ bL_x + n - a = kL_x, \quad k \ge 1 \\ &\Rightarrow\ (b-k)L_x + n = a \\ &\Rightarrow\ x^{(b-k)L_x+n} = x^a \in G_x, \quad k \ge 1. \end{aligned}$$

This is a contradiction to the minimality of b. So, $bL_x + n \le a$. Now suppose that $x^{x-cL_x} = x^a \in G_x$, with c maximal, and suppose that $n - cL_x > a'$. We argue as above:

$$\begin{aligned} &L_x | n - cL_x - a' \\ &\Rightarrow\ n - (k+c)L_x = a', \quad \text{for some } k \ge 1 \\ &\Rightarrow\ x^{n-(k+c)L_x} = x^{a'} \in G_x, \end{aligned}$$

which is a contradiction to the maximality of c. Thus, $n - cL_x \le a'$. □

Theorem 2. *Let S be a semigroup, $x \in S$ a torsion element, and $y \in \langle x \rangle$ any element. Assume the cycle length L_x and cycle start s_x of x are known. Then Algorithm 5 returns the correct values of the discrete logarithm $m = \log_x(y)$ in $O(\sqrt{L_x} + (\log N_x)^2)$ semigroup multiplications, with a required storage of $O(\sqrt{L_x})$ semigroup elements.*

Proof. We use the notations of Algorithm 5 and also write $n = \log_x y$. We will show that the output m is equal to the correct discrete logarithm value n. Recall that we have a group G_x, generated by $x' := x^{tL_x+1}$, and with identity x^{tL_x}. The parameter t is given by the formula $t = \left\lceil \frac{s_x}{L_x} \right\rceil$. Inverses in G_x can be computed in polynomial time using formula (1). Note that membership in G_x can be tested with one equality check: $y \in G_x \Leftrightarrow y \cdot x^{L_x} = y$. There are now two cases:

1. When $y \in G_x$, we have $b = 0$. Here, it is possible to use Shank's baby step–giant step algorithm [5], which is a deterministic algorithm and requires $O(\sqrt{L_x})$ semigroup multiplications and storage space $O(\sqrt{L_x})$, in order to compute $\log_{x'}(y)$. This is done in Step (3). From this value, $n = \log_x(y)$ is readily computed, as shown below. Note that in this case, $\log_x(y)$ is determined modulo L_x.

2. When $y \notin G_x$, Algorithm 5 first computes, using binary search, the smallest power b of x^{L_x} such that the product $y \cdot x^{bL_x}$ lies in the group G_x, and then proceeds as in case 1 via the baby step–giant step algorithm to find the discrete logarithm m' of $y \cdot x^{bL_x}$ with base x' (i.e. $(x')^{m'} = y \cdot x^{bL_x}$). Note that in this case, the value of $\log_x(y)$ is less than s_x, and is thus determined uniquely in $\mathbb{N}$. Again, the time and space complexity are both $O(\sqrt{L_x})$.

In both aforementioned cases, we have the maximal value c such that $x^{m'(tL_x+1)-cL_x} \in G_x$, and so $c \leq L_x + s_x + 1 = N_x + 1$, since $m' \leq L_x$ and $tL_x \leq L_x + s_x$. We also clearly have $b \leq t \leq N_x$. Since the computations of both b and c are done via binary searches, they contribute $O((\log N_x)^2)$ steps to the overall time complexity. Now,

$$x^{m'(tL_x+1)-cL_x} = x^{m'(tL_x+1)} = (x')^{m'} = x^{bL_x+n}.$$

Applying Lemma 5 to the above equation, we must have

$$m'(tL_x + 1) - cL_x \leq bL_x + n, \quad \text{and} \quad bL_x + n \leq m'(tL_x + 1) - cL_x.$$

Therefore, $bL_x + n = m'(tL_x + 1) - cL_x$, or $n = m'(tL_x + 1) - (b + c)L_x$, which is precisely equal to m, the value returned by Algorithm 5. Thus, $m = n$. This completes the proof. □

Combining Theorem 1, Lemma 3, and Theorem 2 we arrive at the main proposition of the article:

Proposition 1. *Let S be a semigroup, $x \in S$ a torsion element, and $y \in \langle x \rangle$ any element. The discrete logarithm $m = \log_x(y)$ can be computed deterministically in*

$$O(\sqrt{N_x} \cdot (\log N_x)^2)$$

steps, with a required storage of $O(\sqrt{N_x})$ semigroup elements.

Proof. For the solution, one begins by finding L_x. This can be done using Algorithm 4 and according to Theorem 1 this requires $O(\sqrt{N_x} \cdot (\log N_x)^2)$ steps and the storage of $O(\sqrt{N_x})$ elements.

By Lemma 3 the computation of the cycle start s_x is achieved in $O((\log N_x)^2)$ semigroup multiplications, which does not contribute to the overall cost of the algorithm.

By Theorem 2, the discrete logarithm m can then be retrieved using Algorithm 5, in $O((\log N_x)^2 + \sqrt{L_x})$ steps, with a required storage of $O(\sqrt{L_x})$ semigroup elements.

As $L_x \leq N_x$, the overall complexity is dominated by the computation of the cycle length, and the proof of the result is now clear. □

4.3 Solving the DLP once the factorization of the cycle length is known

We mentioned in the introduction that for a general group of order N the best general known algorithms for solving the DLP have complexity $O(\sqrt{N})$ operations.

In case the order N has a prime factorization into small primes there is the famous Pohlig–Hellman algorithm [17] for solving the DLP whose complexity is dominated by the largest prime factor in the integer factorization of N.

In case that we have available the integer factorization of the cycle length L_x we can adapt the Pohlig–Hellman algorithm for groups to a Pohlig–Hellman algorithm for solving the DLP in a semigroup. Algorithm 6 represents this adapted Pohlig–Hellman algorithm.

Algorithm 6: Pohlig–Hellman algorithm for solving the DLP in a semigroup

Input A semigroup S, a torsion element $x \in S$, with cycle length $L_x = \prod_{i=1}^{r} p_i^{e_i}$ and cycle start s_x, and $y \in S$ with $y = x^m$

Output The discrete logarithm m of y with base x

1: Compute $t = \left\lceil \frac{s_x}{L_x} \right\rceil$ and define $x' = x^{tL_x+1} \in G_x$.

2: Find the minimum number $0 \leq b \leq t$ such that $y' = y \cdot x^{bL_x} \in G_x$ using binary search.

3: **for** $i \in \{1, \ldots, r\}$

1. Compute the values $x_i' = (x')^{L_x/p_i^{e_i}}$, $y_i' = (y')^{L_x/p_i^{e_i}}$, and $\gamma_i \coloneqq (x_i')^{p^{e_i-1}}$.
2. Calculate the inverse z_i of x_i' in G_x using (1).
3. Set $k \leftarrow 0$ and $n_0 \leftarrow 0$.
4. **while** $k < e_i$ **do**
   (a) Compute $y_k' = (y_i' z_i^{n_k})^{p^{e_i-1-k}} \in \langle \gamma_i \rangle$.
   (b) Use Shank's baby step-giant step algorithm for the group $\langle \gamma_i \rangle \subseteq G_x$ to compute $d_k \in \{0, 1, \ldots, p_i - 1\}$ such that $\gamma_i^{d_k} = y_k'$.
   (c) Set $n_{k+1} \leftarrow n_k + p_i^k d_k$, and $k \leftarrow k + 1$.
5. **end while**
6. Set $m_i \coloneqq n_{e_i}$.

4: **end for**

5: Use the Chinese Remainder Theorem to solve the congruence equations

$$m' \equiv m_i \pmod{p_i^{e_i}}, \quad \forall i \in \{1, \ldots, r\}$$

uniquely for $m' \bmod L_x$. This gives the discrete logarithm of y' with respect to the base x' in the group G_x.

6: Find the maximum number $c \geq 0$ such that $x^{(tL_x+1)m'-cL_x} \in G_x$ using binary search.

7: Return $m = m'(tL_x + 1) - (b + c)L_x$.

Theorem 3. *Let S be a semigroup, $x \in S$ a torsion element, and $y \in \langle x \rangle$ any element. Assume the cycle start s_x of x is known and assume the integer factorization of the cycle length L_x is known to be $L_x = \prod_{i=1}^{r} p_i^{e_i}$. Then Algorithm 6 computes the discrete logarithm $\log_x y$ requiring $O(\sum_{i=1}^{r} e_i(\log L_x + \sqrt{p_i}) + (\log N_x)^2)$ steps. The space complexity of the algorithm consists in $O(\sum_{i=1}^{r} e_i \sqrt{p_i})$ semigroup elements.*

Proof. Steps (1) and (2) are in analogy to the corresponding steps of Algorithm 5. Steps (3)–(5) represent the Pohlig–Hellman algorithm for groups with the implied complexity dominated by the largest prime factor p_i of the integer factorization of L_x (for a reference on Pohlig–Hellman in groups, see in [20, Theorem 2.32]). It follows that the running time of the algorithm is $O(\sum_{i=1}^{r} e_i(\log L_x + \sqrt{p_i}))$ steps. The computation of b and c requires in addition $(\log N_x)^2$ steps. The total space complexity is $O(\sum_{i=1}^{r} e_i \sqrt{p_i})$ semigroup elements and that completes the proof. □

5 Conclusion

The DLP in a finite group has noteworthy significance for cryptography, and so an extension of existing solutions to other algebraic structures like semigroups, where inverses may not be available, is of natural interest. In particular, the DLP in a semigroup has been discussed before in two places, namely refs [12] and

[11]. Both these authors provide probabilistic generalizations of existing collision-based methods for the case of a semigroup. The time complexity of the algorithm in ref. [12] is $O(\sqrt{N_x}(\log N_x)^2 \log\log N_x)$, and the one in ref. [11] is $O(\sqrt{N_x}(\log N_x)^2)$. Both these methods rely on computing a multiple of the cycle length and then taking gcd's or factors and could fail with a small probability that depends on the parameters chosen. In this article, we provided a deterministic solution of the semigroup DLP, which computes the cycle length directly and does not rely on finding a factor of it. The time complexity of our algorithm is $O(\sqrt{N_x}(\log N_x)^2)$. We further demonstrated the application of the Pohlig–Hellman algorithm to semigroups. A direct consequence of our findings is that for cryptographic purposes, generalizing the type of algebraic structure for the DLP offers no additional advantage, at least in the torsion case, both in a classical and a quantum setting.

Funding information: This research was supported by armasuisse Science and Technology. Joachim Rosenthal is also supported by Swiss National Science Foundation grant no. 188430.

Conflict of interest: The authors state no conflict of interest.

References

[1] Diffie W, Hellman ME. New directions in cryptography. IEEE Trans Inform Theory 1976;IT-22(6):644–54.
[2] Menezes AJ, van Oorschot PC, Vanstone SA. Handbook of applied cryptography. CRC Press Series on Discrete Mathematics and its Applications. Boca Raton, FL: CRC Press; 1997.
[3] Cohen H, Frey G, Avanzi R, Doche C, Lange T, Nguyen K, et al., editors. Handbook of elliptic and hyperelliptic curve cryptography. In: Discrete Mathematics and its Applications (Boca Raton). Boca Raton, FL: Chapman and Hall/CRC; 2006.
[4] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. Cambridge, MA: MIT Press; 2009.
[5] Shanks D. Class number, a theory of factorization, and genera. In: Proc. of Symp. Math. Soc., 1971. vol. 20, 1971. pp. 41–440.
[6] Pollard JM. Monte Carlo methods for index computation. Math Comput. 1978;32(143):918–24.
[7] Adleman LM, DeMarrais J. A subexponential algorithm for discrete logarithms over all finite fields. Math Comp. 1993;61(203):1–15.
[8] Kahrobaei D, Koupparis C, Shpilrain V. Public key exchange using matrices over group rings. Groups Complex Cryptol. 2013;5(1):97–115.
[9] Maze G, Monico C, Rosenthal J. Public key cryptography based on semigroup actions. Adv Math Commun. 2007;1(4):489–507.
[10] Goel N, Gupta I, Dass BK. Survey on SAP and its application in public-key cryptography. J Math Cryptol. 2020;14(1):144–52.
[11] Monico C. Semirings and semigroup actions in public-key cryptography. PhD thesis. University of Notre Dame Notre Dame; 2002.
[12] Banin M, Tsaban B. A reduction of semigroup DLP to classic DLP. Des Codes Cryptograph. October 2016;81(1):75–82.
[13] Childs AM, Ivanyos G. Quantum computation of discrete logarithms in semigroups. J Math Cryptol. 01 Dec 2014;8(4):405–16.
[14] Miller GL. Riemannas hypothesis and tests for primality. J Comput Sys Sci. 1976;13(3):300–17.
[15] Rabin MO. Probabilistic algorithm for testing primality. J Number Theory. 1980;12(1):128–38.
[16] Agrawal M, Kayal N, Saxena N. PRIMES is in P. Ann Math (2) 2004;160(2):781–93.
[17] Pohlig S, Hellman M. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). IEEE Trans Inform Theory. 1978;24(1):106–10.
[18] Sutherland AV. Order computations in generic groups. PhD Thesis. Massachusetts Institute of Technology, 2007.
[19] Zumbrägel J. Classification of finite congruence-simple semirings with zero. J Algebra Appl. 2008;7(3):363–77.
[20] Hoffstein J, Pipher J, Silverman JH. An introduction to mathematical cryptography. vol. 1. New York, NY: Springer; 2008.