Math-Net.Ru

Общероссийский математический портал

А. Ю. Nesterenko, Несколько замечаний о задаче дискретного логарифмирования на эллиптических кривых, *Матем. вопр. криптогр.*, 2016, том 7, выпуск 2, 115–120

DOI: <https://doi.org/10.4213/mvk189>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением

<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 54.197.217.227

26 апреля 2023 г., 17:11:32

**Some remarks
on the elliptic curve discrete logarithm problem**

A. Yu. Nesterenko

National Research University Higher School of Economics, Moscow

Получено 03.III.2015

Abstract. We propose an algorithm for solving the discrete logarithm problem on the elliptic curve. This algorithm uses additional information on the multiplicative order of the solution and may be realised in parallel.

Keywords: elliptic curves over the finite field, discrete logarithm problem

**Несколько замечаний о задаче
дискретного логарифмирования на эллиптических кривых**

А. Ю. Нестеренко

Национальный исследовательский университет Высшая школа экономики, Москва

Аннотация. Предлагается алгоритм решения задачи дискретного логарифмирования на эллиптической кривой над конечным полем, использующий дополнительную информацию о мультипликативном порядке искомого решения и допускающий параллельную модификацию.

Ключевые слова: эллиптические кривые над конечным полем, дискретное логарифмирование

Let $p > 3$ be a prime and E be an elliptic curve defined over field $\mathbb{F}_p$ in the short Weierstrass form

$$E: v^2 \equiv u^3 + Au + B \pmod{p},$$

where $4A^3 + 27B^2 \not\equiv 0 \pmod{p}$. Let $P \in E$ be a point of a prime order q , which generates a subgroup $G = \langle P \rangle \in E$.

For every point $Q \in G$ we can define an elliptic curve discrete logarithm problem (ECDLP) as the problem of searching an integer $x \in \mathbb{F}_q^*$ such that

$$Q = [x]P = \underbrace{P + \dots + P}_x. \quad (1)$$

We can use Pollard's Rho and Lambda methods or parallel algorithm introduced by van Oorschot and Wiener, see [2, 4], to solve this problem, see [1]. The running time of these algorithms is $O(\sqrt{q})$ and doesn't depend on x .

In this paper we present a new algorithm, based on ideas of Pollard, van Oorschot and Wiener, which has running time dependent on x .

The main result is as follows. If we know a multiplicative order of x in $\mathbb{F}_q^*$, i.e. if we know r such that

$$r \mid q-1 \quad \text{and} \quad x^r \equiv 1 \pmod{q}, \quad (2)$$

then we can find x with running time $O(\sqrt{r} \log q)$.

1. Background

Let g be a primitive root modulo q and $r \mid q-1$. Define

$$a \equiv g^{\frac{q-1}{r}} \pmod{q},$$

where $a^k \not\equiv 1 \pmod{q}$ for any $k \mid r$, $k > 1$.

It's well known that for every x of order r there exists an integer n , $0 < n < r$, such that $x \equiv a^n \pmod{q}$. Hence, we can write

$$Q = [x]P = [a^n]P \quad (3)$$

and now we have to find n .

Define $h = \lceil \sqrt{r} \rceil$ and write $n = n_1 h - n_0$, where $0 \leq n_0 < h$, $0 < n_1 \leq h$. Hence, we can write (3) in the form

$$[(a^h)^{n_1}]P = [a^{n_0}]Q,$$

and using "baby steps, giant steps", see [5], find values of n_0 and n_1 .

Since the evaluation of $[a^\xi]P$ has running time $O(\log q)$ for every $\xi \in \mathbb{F}_q^*$ and values of n_0, n_1 are not larger than $\sqrt{r}$, the running time of "baby steps, giant steps" algorithm is of the order $O(\sqrt{r} \log q)$.

2. The algorithm

Since the “baby steps, giant steps” algorithm uses a large amount of memory, in practical evaluation we can use a modification of Pollard’s Lambda method, see [4, 6]. This modification directly finds a solution of (1).

Let $s = \lceil \log_2 r \rceil$, we choose random numbers $\xi_0, \dots, \xi_{s-1}$ such that $0 < \xi_i < r$, $i = 0, \dots, s-1$. Define a map $f : E \rightarrow E$. For any point $R = (x_R, y_R) \in E$ let

$$f(R) = [\zeta_i] R, \quad \text{where } i \equiv x_R \pmod{s}, \quad \zeta_i \equiv a^{\xi_i} \pmod{q}. \quad (4)$$

On the first step we construct a sequence of points

$$R_{k+1} = f(R_k), \quad R_k \in E, \quad R_0 = P, \quad k = 0, 1, \dots \quad (5)$$

For every point we have

$$R_{k+1} = [\zeta_k] R_k = [\zeta_k \zeta_{k-1}] R_{k-1} = \dots = [\mu_{k+1}] P,$$

where $\mu_{k+1} \equiv \prod_{j=0}^k \zeta_j \pmod{q}$ and ζ_j is defined by (4).

Starting with k_0 , we can keep all points $R_{k_0+1}, \dots, R_{k_0+b}$ for some natural number b along with corresponding values $\mu_{k_0+1}, \dots, \mu_{k_0+b}$. This set S of kept points constitutes a trap-set with cardinality $|S| = b$. The exact values of k_0, b may be found in [6].

On the last step of the algorithm we construct a sequence of points $U_0, U_1, \dots$, starting from a random point $U_0 = [a^\xi] Q$, where ξ is a random integer, $0 < \xi < r$. Our sequence consists of points

$$U_{k+1} = f(U_k), \quad U_k \in E, \quad k = 0, 1, \dots, \quad (6)$$

and every point U_k in this sequence may be represented as

$$U_{k+1} = [\zeta_k] U_k = [\zeta_k \zeta_{k-1}] U_{k-1} = \dots = [\nu_{k+1}] Q,$$

where $\nu_{k+1} \equiv a^\xi \prod_{j=0}^k \zeta_j \pmod{q}$,

For every point U_k , $k = 1, 2, \dots$, we can check a condition $U_k \in S$, where S is a trap-set. If the condition holds, then we find some index j such that

$$[\mu_j] P = R_j = U_k = [\nu_k] Q, \quad (7)$$

hence, $\mu_j \equiv \nu_k x \pmod{q}$ and

$$x \equiv \mu_j \nu_k^{-1} \pmod{q} \quad (8)$$

is a solution of (1). If our condition doesn’t hold for all $U_1, \dots, U_h$, we need to choose another value of ξ and repeat this step again.

It's easy to see that the algorithm runs successfully only if a set of points

$$[a]P, [a^2]P, \dots, [a^{r-1}]P, [a^r]P = P$$

contains the point Q . This is equivalent to the condition (2).

3. Parallel version of the algorithm

Our algorithm may be easily modified for parallel computations. Consider a parallel paradigm of evaluation. Let there are $2w$ threads with common memory and every thread may produce evaluations independently.

We fix an integer b and divide all set of threads in two groups, say $\mathcal{P}$ and $\mathcal{Q}$, with the same cardinality w and the common sets of trap-points, $S_{\mathcal{P}}$ and $S_{\mathcal{Q}}$ correspondingly.

Every thread of the first group $\mathcal{P}$ realizes the following procedure: it chooses a random point $R_0 = [\zeta_0]P$, where $\zeta_0 \equiv a^\xi \pmod{q}$ and ξ is a random integer, $0 < \xi < r$, and constructs a sequence

$$R_k = f(R_{k-1}), \quad R_k = (x_{R,k}, y_{R,k}) \in \mathcal{E}, \quad k = 1, 2, \dots,$$

where map f is defined by (4). As before, for every point we have

$$R_k = [\mu_k]P, \quad \mu_k \equiv \prod_{j=0}^k \zeta_j \pmod{q}.$$

For every point R_k a thread checks the condition $x_{R,k} \equiv 0 \pmod{b}$. If the condition holds, the thread searches the point R_k in $S_{\mathcal{Q}}$ set of trap-points. If the thread doesn't find this point in $S_{\mathcal{Q}}$, then this point is added to the set $S_{\mathcal{P}}$.

In a similar way, every thread in a second group $\mathcal{Q}$ evaluates a sequence

$$U_k = [\nu_k]Q, \quad U_k = (x_{U,k}, y_{U,k}), \quad \nu_k \equiv \prod_{j=0}^k \eta_j \pmod{q},$$

beginning from a random point $U_0 = [a^\xi]Q$ with random integer $0 < \xi < r$.

For every point U_k a thread checks the condition $x_{U,k} \equiv 0 \pmod{b}$. If the condition holds, the thread searches this point in $S_{\mathcal{P}}$ set of trap-points. If the thread doesn't find the point U_k in $S_{\mathcal{P}}$, then this point is added to the set $S_{\mathcal{Q}}$.

In this manner, we can evaluate w sequences of points started from the point $P \in \mathcal{E}$ and w sequences started from the point $Q \in \mathcal{E}$. All trap-points in sets $S_{\mathcal{P}}$ and $S_{\mathcal{Q}}$ satisfy the condition $x \equiv 0 \pmod{b}$, where x is an x -coordinate of point. When the thread finds a trap-point, then we obtain an equality (7) and can find a solution using (8).

As an example, we demonstrate the results of practical implementation of our parallel algorithm. Consider an elliptic curve $\mathcal{E}$ defined by the equation

$$v^2 \equiv u^3 - 2u + 83161154912977162385779023371676872267 \pmod{p},$$

where $p = 170144519623114011343322490539658030031$ is prime and condition $\lceil \log_2 p \rceil = 128$ holds. This curve has an order $2q$, where q is prime and

$$q = 85072259811557005664489355982895124223,$$

$$q - 1 = 2 \cdot 3 \cdot 139 \cdot 643 \cdot 12281 \cdot 51593 \cdot 53887 \cdot 4646248420547414411.$$

We need to find an integer $x \in \mathbb{F}_q^*$, such that $Q = [x]P$, where

$$P = (127079991335379663215392766670928701845,$$

$$146478651337885753760004547813245055780),$$

$$Q = (135768511924514909185266977775889591902,$$

$$104833826712192434308111206165615249692),$$

$|\langle P \rangle| = q$ and $Q \in \langle P \rangle$. Additionally, we know that the multiplicative order of x in $\mathbb{F}_q^*$ are equal to $r = 34143537841471 = 12281 \cdot 51593 \cdot 53887$, $\lceil \log_2 r \rceil = 46$.

Firstly, we find a generator $a \in \mathbb{F}_q^*$, having the order r :

$$a = 69038627934287400758544579548029658020,$$

and construct 46 values $\zeta_1, \dots, \zeta_{46} \in \mathbb{F}_q^*$, where $\zeta_i \equiv a^{\xi_i} \pmod{q}$ are random integers for all $i = 1, \dots, 46$ and $0 < \xi_i < r$. To simplify the evaluation we choose random integers ξ_i with bounded Hamming weights of values ζ_i , say $w(\zeta_i) < 40$, where Hamming weights are defined by

$$w(\zeta_i) = \sum_{j=0}^{127} c_{i,j}, \quad \zeta_i = \sum_{j=0}^{127} c_{i,j} 2^j, \quad c_{i,j} \in \{0, 1\}, \quad i = 1, \dots, 46.$$

This simple trick gives us 25% of speed up.

After this, our personal computer (AMD processor with 8 cores, 2.5 GHz) calculated 8 sequences for $b = 1024$. When we find two points R_j, U_k satisfying equality (7)

$$R_j = [1785741636823453981528591882827425715]P =$$

$$= [3922469994082601799559937328263430847]Q = U_k,$$

for some values of indexes j and k , we immediately calculate

$$x = 11199326890025042093039962745239277210.$$

The running time of all calculations was 224 seconds, the cardinality of $S_{\mathcal{P}}$ was 2270 and cardinality of $S_{\mathcal{Q}}$ was 2356.

4. Conclusion

We describe an algorithm which seems not to be useful for real cryptosystems, because usually the multiplicative order of x is not known. From the other side, the value of q is known. Hence, it is possible to find all factors of $q - 1$ and to check for every factor r whether x has an order r .

If x is a primitive root modulo q , then our algorithm has a maximal complexity $O(\sqrt{q} \log_2 q)$, which is a bit larger than complexity of Pollard and van Oorschot-Wiener methods.

References

- [1] Blake I., Seroussi G., Smart N., *Elliptic Curves in Cryptography*, London Math. Soc. Lect. Notes, **265**, Cambridge: Cambridge Univ. Press, 1999, 224 pp.
- [2] van Oorschot P., Wiener M. J., "Parallel collision search with cryptanalytic applications", *J. Cryptology*, **12**:1 (1999), 1–28.
- [3] Pollard J. M., "Theorems on factorization and primality testing", *Proc. Cambridge Phil. Soc.*, **76**:3 (1974), 521–528.
- [4] Pollard J. M., "Monte Carlo methods for index computation (mod p)", *Math. Comput.*, **32**:143 (1978), 918–924.
- [5] Shanks D., "Class number, a theory of factorization and genera". In: "1969 Number Theory Institute", Proc. Symp. Pure Math., **XX**, 1971, 415–440.
- [6] Teske E., "Square-root algorithms for the discrete logarithm problem (a survey)". In: "Public-key Cryptography and Computational Number Theory", Berlin etc.: Walter de Gruyter, 2001, 283–301.