Solving ECDLP over $F_p$ with Pre-computation via Representation Technique
Claire Delaplace, Pierre-Alain Fouque, Paul Kirchner, Alexander May

To cite this version:
Claire Delaplace, Pierre-Alain Fouque, Paul Kirchner, Alexander May. Solving ECDLP over $F_p$ with Pre-computation via Representation Technique. 2020. hal-02427655

HAL Id: hal-02427655
https://inria.hal.science/hal-02427655
Preprint submitted on 3 Jan 2020
Solving ECDLP over $\mathbb{F}_{p^\ell}$ with Pre-computation via Representation Technique

Claire Delaplace\textsuperscript{1}, Pierre-Alain Fouque\textsuperscript{2}, Paul Kirchner\textsuperscript{2}, and Alexander May\textsuperscript{1}\textsuperscript{*}

\textsuperscript{1} Horst Görtz Institute for IT Security
Ruhr University Bochum, Germany
\textsuperscript{2} Univ Rennes 1, CNRS, IRISA
Rennes, France

Abstract. We present a new algorithm solving the elliptic curve discrete logarithm problem (ECDLP) with pre-computation. Our method is based on recent work from Delaplace and May (NuTMiC 2019) which aims to solve ECDLP over some quadratic field $\mathbb{F}_{p^2}$ using the so-called representation technique. We first revisit Delaplace and May’s algorithm and show that with better routines, it runs in time $p^{3.114}$, improving on the $p^{3.314}$ time complexity given by the authors. Then, we show that this method can be converted to a $p^{4/5}$-algorithm, assuming that we have performed a $p^{6/5}$-pre-computation step in advance. Finally, we extend our method to other fields $\mathbb{F}_{p^\ell}$, where $\ell \geq 2$ is a constant, leading to a $p^{2/(2\ell+1)}$-algorithm, after a $p^{(\ell+1)/(2\ell+1)}$-pre-computation step. Although our method does not improve on previous ECDLP with pre-computation algorithm (namely the one from Bernstein and Lange), we think it offers an elegant alternative, which can hopefully lead to further improvements, the idea of using the representation technique for ECDLP being very new.

1 Introduction

Elliptic curves are important tools of today’s cryptography. In particular, the \textit{elliptic curve discrete logarithm problem (ECDLP)} is one of the key challenges of current public key cryptography. Given an elliptic curve $E$ defined over a finite field $\mathbb{F}_q$ and two points $P$ and $Q$ on this curve, this problem consists in recovering some integer $k$ such that $kP = Q$.

This problem has been widely studied by the community in the last fifty years (e.g. [Sha71,Sem98,Gau09,GG16,PKM16,GWZ17]). We refer the reader to [GG16] for a general overview of existing ECDLP algorithms.

For elliptic curves defined over $\mathbb{F}_p$ where $p$ is a prime, \textit{Pollard’s Rho} algorithm obtains the best time and memory complexity, requiring $O(\sqrt{p})$ operations over $\mathbb{F}_p$, and constant storage. This method basically works by creating a sequence $(R_i)_{i \geq 1}$ of elements of $(P)$ with $R_0 = O$, the point at infinity of the elliptic

\textsuperscript{*} Funded by DFG under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972.
curve, and \( R_{i+1} = g(R_i) \) for a well chosen function \( g \), such that each \( R_i \) satisfies \( R_i = a_i P + b_i Q \) for some \( a_i \) and \( b_i \) in \( \mathbb{Z} \). When a collision appears between two elements \( R_i \) and \( R_j \), we have \((a_i - a_j)P = (b_j - b_i)Q\). Assuming that \( b_i \neq b_j \), this leads to \( k = \frac{a_i - a_j}{b_j - b_i} \), being the solution.

Another interesting algorithm due to Pollard is the so-called kangaroo method. The algorithm computes two sequences \((P_i)\) (the “tamed kangaroo”) and \((R_i)\) (the “wild kangaroo”) such that each \( P_i \) satisfies \( P_i = a_i P \) for some \( a_i \) in \( \mathbb{Z} \) and each \( R_i \) satisfies \( R_i = Q - b_i P \). When a collision appears between two elements \( P_i \) and \( R_j \) of these sequences, we have \( a_i P = Q - b_i P \) leading to \( k = a_i + b_i \) being the solution. All in all this procedure uses \( \mathcal{O}(\sqrt{p}) \) operations over \( \mathbb{F}_p \) and requires to store \( \mathcal{O}(\sqrt{p}) \) elements of \( \mathbb{F}_p \). Many variants exists in various settings (e.g. [VOW99,FJM14,BL12,BL13]). In this paper, we are particularly interested in the one of [BL12,BL13] proposed by Bernstein and Lange, which studies the case of solving \((\text{EC})\text{DLP}\) with pre-computation.

In this setting, we assume that the elliptic curve \( \mathbb{E} \) is given along with a generator \( P \) of a (large) subgroup of the elliptic group. This procedures recovers \( k \) such that \( Q = kP \), for any input point \( Q \) in the group \( \mathbb{G} \) generated by \( P \). Note that here, we assume that the point \( P \) is independent of the instance. If we want to find \( k \) such that \( Q = kP' \) for a different generator \( P' \) belonging to \( \mathbb{G} \), we can find \( k_1 \) and \( k_2 \) such that \( Q = k_1 P, P' = k_2 P \). The solution \( k \) would then be such that \( k \cdot k_2 P = k_1 P \). In other words \( k = k_1k_2^{-1} \mod r \), where \( r \) is the (well known) order of \( P \). This does not change the overall complexity of the algorithm.

The idea of [BL12,BL13] is to pre-compute \( p^\lambda \) “tamed kangaroos” sequences with different starting points and terminate them when a so called distinguished point is reached (c.f. [VOW99]). For each sequence, only the first and the last points are stored. The algorithm then works by computing the “wild kangaroo” sequence \((R_i)\), until a distinguished point \( D \) is reached. If this \( D \) is also the ending point of one of the “tamed kangaroos” sequence \((P_i)\), then a collision can be found by re-evaluating \((P_i)\) and \((R_i)\). Otherwise the procedure is restarted with a different starting point for the “wild kangaroo” sequence. All in all this procedure requires to perform \( \mathcal{O}(p^{1-\lambda}) \) operations over \( \mathbb{F}_p \), and to store \( \mathcal{O}(p^{1-\lambda}) \) elements of \( \mathbb{F}_p \). The best time and memory complexity given is \( \mathcal{O}(p^{1/3}) \), after a pre-computation which costs \( \mathcal{O}(p^{2/3}) \).

For elliptic curves defined over \( \mathbb{F}_{p^\ell} \) for some constant \( \ell \leq 2 \) and \( p \) being a prime or a power of a prime, it is possible to solve the problem using index calculus type of algorithms. This particular case has been widely studied by the community (e.g. [GS99,GHS02b,GHS02a,Gau09] ) The idea is then to split a point into an \( \ell \)-sum of points from a factor basis that is defined over the field \( \mathbb{F}_p \). Relying on Weil’s descent and Semaev’s summation polynomials [Sem04], this factor basis can be computed via a Gröbner basis computation. This yields a relation between factor basis elements. If at least \( q \) – the size of the factor base – such relations are collected, ECDLP can be solved via linear algebra. In total, Gaudry’s method [Gau09] runs in time \( \mathcal{O}(p^{2-\ell}) \). For \( \ell = 2 \) this is the
same complexity as Pollard’s Rho algorithm. These methods seem quite hard to transpose to the setting where \( p \) is small (e.g. \( p = 2 \)) and \( \ell \) is a large prime extension due to the fact that some \( \ell \) factor is hidden in the \( O() \). However the idea of using index calculus based method to improve algorithms for \( \text{ECDLP} \) in characteristic 2 has also been investigated (e.g. [FPPR12,PQ12,HPST13,GG14]).

Recently, Delaplace and May [DM19] proposed another method to solve \( \text{ECDLP} \) with elliptic curves defined over a field \( \mathbb{F}_p^2 \). Their algorithm relies on the representation technique introduced by Howgrave-Graham and Joux [HGJ10] to solve the subset-sum problem. In this particular context, the representation technique can break below the square-root barrier [HGJ10,BCJ11]. Unfortunately, the results of [DM19] were not as successful. The idea behind [DM19] is that the solution \( k \) can be split as the sum \( k = k_1 + k_2 \) for \( k_1 \) and \( k_2 \) having a specific “good shape” in \( p \) different ways, each of these \( p \) representation \((k_1, k_2)\) of \( k \) leading to the solution. The idea is then to compute only a \( p^{-1} \) fraction of all possible \( k_1P \) and \( Q - k_2P \) (e.g. points \((x, y)\) such that \( x \in \mathbb{F}_p \)). There should still be a pair \((k_1, k_2)\) satisfying \( k_1P = Q - k_2P \) which survives. In order to do so the authors of [DM19] split again \( k_1 \) into the sum \( k_{11} + k_{12} \) and \( k_2 \) into \( k_{21} + k_{22} \), for \( k_i \) of “good shape” coming up with a 4-list algorithm. Indeed the authors of [DM19] first build lists \( L_{11} = \{k_{11}P\}, L_{12} = \{k_{12}P\}, L_{21} = \{Q - k_{21}\} \) and \( L_{22} = \{-k_{22}\} \). Then, they construct the lists \( L \) and \( L' \) such that \( L = \{(x, y) \in L_{11} + L_{12}, x \in \mathbb{F}_p\} \) and \( L' = \{(x, y) \in L_{21} + L_{22}, x \in \mathbb{F}_p\} \), and search for a collision between these two lists. In order to perform the second step, they define a “join” problem, which given two lists \( L_1 \) and \( L_2 \) of points of \( E(\mathbb{F}_p^2) \) such that \(|L_1||L_2| = p^{3/2}\) consists in returning a list \( L \) of all \((x, y) = P_1 + P_2\) with \( P_1 \in L_1, P_2 \in L_2\) such that \( x \in \mathbb{F}_p \). They show that if this problem can be solved in time \( T \) and in memory \( M \), then one can also solve \( \text{ECDLP} \) over \( \mathbb{F}_p^2 \) in time \( T \) and memory \( M \). Delaplace and May proposed a way to solve this problem, relying on multipoint-evaluation of bivariate polynomials. Using results from [NZ04] they obtained a \( O(p^{1.314}) \) algorithm.

**Our contributions.** We extend the work of Delaplace and May in several ways. First we present a \( \tilde{O}(p^{1+\epsilon}) \)-algorithm for the join problem defined in [DM19], relying on Kedlaya and Umans multivariate multipoint evaluation algorithm [KU08] rather than Nüsken and Ziegler. This reduces the time complexity of Delaplace and May’s algorithm.

Second, we show that with a few modifications, this algorithm can be turned to an algorithm solving \( \text{ECDLP} \) for an elliptic curve defined over \( \mathbb{F}_p^2 \) in time and memory \( \tilde{O}(p^{c+\epsilon}) \), for any constant \( \epsilon > 0 \), and for any \( 4/5 \leq c \leq 1 \) after a \( \tilde{O}(p^{2-c+\epsilon}) \)-pre-computation step. The core idea is that the lists \( L_{11}, L_{12}, L_{22} \) as well as the join \( L \) of \( L_{11} \) and \( L_{12} \) do not depend on the target \( Q \) and can then be precomputed in advance. For values \( c \) smaller than \( 4/5 \), the same algorithm can still be used with a larger memory consumption as the storage of the list \( L \) will dominate the memory complexity. The question of reducing the memory consumption in this case without increasing the runtime remains as an open problem.
Finally we extend this method to other finite fields $\mathbb{F}_{p^\ell}$, for $\ell \geq 2$ a constant. We obtain a $\tilde{O}(p^{\ell+c})$ in time and space algorithm, for any constant $c > 0$, and for any $\ell^2/(2\ell + 1) \leq c \leq \ell/2$ after a $\tilde{O}(p^{\ell-c+\epsilon})$-pre-computation step. Once again, we could potentially use smaller values of $c$, but the memory consumption would then increase due to the space required to store the join list $L$.

Although our new algorithm does not outperform the one from Bernstein and Lange, we think it offers a refreshing alternative. Furthermore, it gives us reason to think that the use of representation technique for ECDLP is worth further investigation. Hopefully the results presented here can still be improved since the idea of using representation technique for ECDLP is genuinely new.

Organisation of the paper. After recalling the important notions and properties in Section 2, we revisit and improve the algorithm from [DM19] in Section 3. Then we propose our new ECDLP algorithm with pre-computation for curves over $\mathbb{F}_{p^\ell}$ in Section 4. Finally we extend this result to other fields $\mathbb{F}_{p^\ell}$, for a constant $\ell \geq 2$ and discuss the limitations and possible improvements in Section 5.

2 Preliminaries

For any integer $r$, we denote by $\mathbb{Z}_r$ the ring $\mathbb{Z}/r\mathbb{Z}$. For any prime or power of a prime $q$, we denote by $\mathbb{F}_q$ the finite field with $q$ elements. Let $\ell$ be a constant, every $x \in \mathbb{F}_{p^\ell}$ can uniquely be expressed as $x = x_0 + \alpha x_1 + \ldots + \alpha^{\ell-1} x_{\ell-1}$ where $\alpha$ is a symbolic root of an irreducible polynomial $P$ of degree $\ell$ over $\mathbb{F}_p$. The tuple $(1, \alpha, \ldots, \alpha^{\ell-1})$ is a basis of $\mathbb{F}_{p^\ell}$ seen as an $\mathbb{F}_p$-vector space of dimension $\ell$. For any $x > 0$, we denote by $\log(x)$ the logarithm of $x$ in basis 2. Let $(G, +)$ be an abelian group and let $S_1$ and $S_2$ be two sets of elements of $G$. We denote by $S_1 \cap S_2$ the set of all $x \in G$ such that $x = x_1 + x_2$ with $x_1 \in S_1$ and $x_2 \in S_2$. Let $L = (\ell_0, \ldots, \ell_{N-1})$ be a list. We denote by $|L| = N$ the size of $L$ and for all $0 \leq i < N$, we denote the $(i+1)$-th entry of $L$ by $L[i] = \ell_i$. If $L_1 = (\ell_0, \ldots, \ell_{N-1})$ and $L_2 = (t_0, \ldots, t_{M-1})$ are two lists whose entries belong to an abelian group, we denote by $L_1 + L_2$ the list $(\ell_0 + t_0, \ldots, \ell_0 + t_{M-1}, \ldots, \ell_{N-1} + t_0, \ldots, \ell_{N-1} + t_{M-1})$. An empty list is represented by the symbol $\bot$. Let $f_0, \ldots, f_{N-1}$ be a set of polynomials in $\mathbb{F}_p[X_1, \ldots, X_m]$. We denote by $\langle f_0, \ldots, f_{N-1} \rangle$ the ideal spanned by $f_0, \ldots, f_{N-1}$.

Elliptic curves and discrete logarithm problem. Let $p > 3$ be a prime, and let $q = p^\ell$. Let $E$ be an elliptic curve over $\mathbb{F}_p$. We denote by $O$ the point at infinity of $E$, and by $E(\mathbb{F}_q)$ the group formed by all the points of $E$, called the elliptic group. Similar to [DM19] in this paper we represent elliptic curves using Weierstraß’s equations. In other words, let $f : \mathbb{F}_q \to \mathbb{F}_q$ be such that $f(x) = x^3 + ax + b$ for some $a, b \in \mathbb{F}_q$. We have $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q^2 : y^2 = f(x)\} \cup \{O\}$. We denote by $|E(\mathbb{F}_q)|$ the order of the elliptic group. By Hasse’s theorem, we now that $|E(\mathbb{F}_q)| = \Theta(q)$. Let $P \neq O$ be a point of $E(\mathbb{F}_q)$. We denote by $\langle P \rangle$ the subgroup of $E(\mathbb{F}_q)$ which is generated by $P$, that is $\langle P \rangle$ is the set of all $Q$
such that there exists \( k \in \mathbb{Z} \) satisfying \( Q = kP \). We denote by \( r \) the order of \( P \), which is also the number of elements in \( \langle P \rangle \).

**Definition 1 (ECDLP).** Let \( \mathbb{F}_q \) be a finite field and let \( E \) be an elliptic curve over \( \mathbb{F}_q \). Let \( G \) be a large subgroup (possibly equal to) of prime order of \( E(\mathbb{F}_q) \), and let \( P \) be a generator of \( G \). We denote by \( r = \mathcal{O}(q) \) the order of \( G \). Given a point \( Q \in G \), solving the discrete logarithm problem over \( G \) consists in recovering \( k \in \mathbb{Z}_r \) such that \( kP = Q \).

Similar to [DM19] we also call the particular case where \( q = p^2 \) where \( p \) is a prime the \( p^2\text{-ECDLP} \) problem.

The complexity analyses of the algorithms we present later rely on the following heuristic which was already stated in [DM19]. Some theoretical justification can be found in recent work by Kim and Tibouchi [KT19].

**Heuristic 1.** Let \( E \) be an elliptic curve over a finite field \( \mathbb{F}_q \). The \( x \)-coordinate of the points \( P \neq O \) of \( E(\mathbb{F}_q) \) are uniformly distributed in \( \mathbb{F}_q \).

**Complexities.** We use the standard Landau notations to describe the complexities presented in this paper. All time complexities are given in number of elementary operations (negations, additions, multiplications, inverse) over \( \mathbb{F}_p \), and the memory complexity is the number of elements of \( \mathbb{F}_p \) that we need to store. In particular, this means that for any constant \( \ell \), all elementary operations over \( \mathbb{F}_{p^\ell} \) and the sum of two points in \( E(\mathbb{F}_{p^\ell}) \) can be performed in time \( \mathcal{O}(1) \), and storing an element of \( \mathbb{F}_{p^\ell} \) or a point of \( E(\mathbb{F}_{p^\ell}) \) requires \( \mathcal{O}(1) \) memory. We also use the \( \tilde{\mathcal{O}}(\cdot) \) notation, to hide poly-logarithmic factors in \( p \). In other words, \( \tilde{\mathcal{O}}(X) = \mathcal{O}(X (\log p)^c) \), where \( c \) is a constant.

**Representation technique.** The representation technique was introduced in [HGJ10] by Howgrave-Graham and Joux to solve the subset-sum problem. In this context, given a vector \( a \in \mathbb{Z}^n \), and a target \( t \in \mathbb{Z} \), the goal is to recover a vector \( e \in \{0,1\}^n \), so that the dot product between \( a \) and \( e \) satisfies \( \langle a, e \rangle = t \). Howgrave-Graham and Joux noticed that if \( e \) consists of exactly \( n/2 \) ones, then \( e \) can be represented as the sum of two vectors \( e_1 + e_2 \) such that \( e_1 \) and \( e_2 \) consist of \( n/4 \) ones in \( \binom{n/2}{n/4} \) ways. Finding only one of these representations \( (e_1, e_2) \) of \( e \) is all that is needed to solve the problem. As such, additional constraints can be enforced on the dot products \( \langle a, e_1 \rangle \) and \( \langle a, e_2 \rangle \) so that only one of these representations survives with high probability. This yields to the first subset-sum algorithm breaking below the square-root bound with a complexity of \( 2^{0.337n} \).

In [DM19], Delaplace and May investigated whether a similar method can be used to solve the \( p^2\text{-ECDLP} \) problem. We detail and revisit their idea in Section 3.

For now, we recall the definition of representation as it was given in [DM19].

**Definition 2.** Let \( t > 0 \), \( r \in \mathbb{N} \) and \( k \in \mathbb{Z}_r \). Let \( S_1, \ldots, S_t \) be subsets of \( \mathbb{N} \). A tuple \( (k_1, \ldots, k_t) \in S_1 \times \cdots \times S_t \) is a representation of \( k \) if \( k = k_1 + \cdots + k_t \) (mod \( r \)).
Useful results. Here, we recall some lemmas which we utilise to support our complexity claims. The first one is the Schwartz-Zippel Lemma, which estimates the probability that a random point in $\mathbb{F}_p^m$ is a zero of a given polynomial in $m$ variables.

**Lemma 1 (Schwartz-Zippel Lemma).** Let $F \in \mathbb{F}_p[X_1, \ldots, X_m]$ be a polynomial of total degree at most $d$. Let $(x_1, \ldots, x_m)$ be a randomly chosen point in $\mathbb{F}_p^m$, the probability that $(x_1, \ldots, x_m)$ is a zero of $F$ satisfies

$$P[F(x_1, \ldots, x_m) = 0] \leq \frac{d}{p}.$$ 

We also rely on an algorithm due to Kedlaya and Umans [KU08] which evaluates simultaneously a polynomial $F \in \mathbb{F}_p[X_1, \ldots, X_m]$ in $N$ points of $\mathbb{F}_p^m$. The bit complexity of this algorithm was originally studied in [KU08]. These results have recently been refined and extended to other computational models (namely Turing machine) by Van der Hoeven and Lecerf [vdHL19]. In our complexity model however, the difference between the analysis of [KU08] and [vdHL19] disappears in the $\tilde{O}(\cdot)$.

**Lemma 2 (Kedlaya-Umans [KU08]).** Let $F$ be a polynomial in $m$ variables over a prime field $\mathbb{F}_p$, with individual degree less than $d$. For any fixed constant $\epsilon$, it is possible to evaluate simultaneously $F$ in $N$ points of $\mathbb{F}_p^m$ in time $\tilde{O}(\ell^{d m + N})^{1+\epsilon}$.

Finally we also rely on Weil-descent to transform a polynomial equations $g(x_1, \ldots, x_n) = 0$ over $\mathbb{F}_p^\ell$ to a system of $\ell$ polynomial equations over $\mathbb{F}_p$.

**Lemma 3 (Weil descent, stated as in [GG16] lemma 1).** Let $(1, \alpha, \ldots, \alpha^{\ell-1})$ be a basis of $\mathbb{F}_p^\ell$ seen as a vector space over $\mathbb{F}_p$. Let $g(x_1, \ldots, x_m) \in \mathbb{F}_p[X_1, \ldots, X_m]$. We denote by $y$ the tuple $y = (y_{1,0}, \ldots, y_{1,\ell-1}, \ldots, y_{m,0}, \ldots, y_{m,\ell-1})$. Then there exist unique polynomials $g_i(y) \in \mathbb{F}_p[y], 0 \leq i < \ell$ such that

$$f(y_{1,0} + \cdots + \alpha^{\ell-1}y_{1,\ell-1}, \ldots, y_{m,0} + \cdots + \alpha^{\ell-1}y_{m,\ell-1}) = \sum_{i=0}^{\ell-1} \alpha^i f_i(y).$$

Furthermore if $f(a_1, \ldots, a_m) = 0$, for some $a_1, \ldots, a_m \in \mathbb{F}_p^\ell$, there exist $b_{k,j} \in \mathbb{F}_p, 1 \leq k \leq m, 0 \leq j < \ell$, such that $a_k = \sum_{j=0}^{\ell-1} b_{k,j} \alpha^j$ and for all $i$, $f_i(b) = 0$, where $b = (b_{1,0}, \ldots, b_{1,\ell-1}, \ldots, b_{m,0}, \ldots, b_{m,\ell-1})$.

### 3 Revisiting Delaplace and May Algorithm

We focus here on solving the $p^2$-ECDLP problem. We consider an elliptic curve $E$ given by its Weierstrass equation, a large subgroup $G$ of $E(\mathbb{F}_p^2)$ of prime order $r$ generated by a point $P$ and a target point $Q \in G$ such that $Q = kP$, for some unknown $k \in \mathbb{Z}_r$. Let $(1, \alpha)$ be a base of $\mathbb{F}_p^2$ seen as a $\mathbb{F}_p$-vector space.
3.1 General Idea

In [DM19] Delaplace and May proposed a new algorithm using representation technique. The main idea is that the solution $k$ can be represented by a pair $(k_1, k_2)$ such that $k_1$ (resp. $k_2$) belongs to a certain well chosen subset $S_1$ (resp. $S_2$) of $\mathbb{N}$, and $(k_1, k_2)$ satisfies $k = k_1 + k_2 \mod r$. The sets $S_1$ and $S_2$ are constructed so that there are at least $r/p \approx p$ representations $(k_1, k_2)$ of $k$. It is then possible to reduce the search space by computing only those which satisfy a certain criteria, which is verified by a $p^{-1}$ fraction of all the points of $E(\mathbb{F}_p^2)$. Their algorithm is recalled in Algorithm 1 and illustrated in Figure 1.

To summarise, the authors of [DM19] did the following. They computed the lists $L$ and $L'$ such that

$$L = \{k_1P = (x, y) : k_1 \in S_1, \ x \in \mathbb{F}_p\},$$
$$L' = \{Q - k_2P = (x', y') : k_2 \in S_2, \ x' \in \mathbb{F}_p\},$$

and searched for a collision $k_1P = Q - k_2P$ between $L$ and $L'$. They argued that, as long as $p$ is large enough, we can expect one collision to exist.

The difficulty lies in building the lists $L$ and $L'$ efficiently. Delaplace and May proposed the following 4-list method. Consider that $k_1$ is the sum $k_{11} + k_{12}$ for a unique pair $(k_{11}, k_{12}) \in S_{11} \times S_{12}$ where $S_{11}$ and $S_{12}$ are well chosen subsets of $S_1$ such that $S_{11} + S_{12} = S_1$. Accordingly, $k_2$ is split uniquely as $k_2 = k_{21} + k_{22}$, with $(k_{21}, k_{22}) \in S_{21} \times S_{22}$, where $S_{21}$ and $S_{22}$ are two subsets of $S_2$ satisfying $S_{21} + S_{22} = S_2$. The algorithm then works by building base lists $L_{11}$, $L_{12}$, $L_{21}$ and $L_{22}$ such that

$$L_{11} = \{k_{11}P : k_{12} \in S_{11}\}$$
$$L_{12} = \{k_{12}P : k_{12} \in S_{12}\}$$
$$L_{21} = \{Q - k_{21}P : k_{21} \in S_{21}\}$$
$$L_{22} = \{-k_{22}P : k_{22} \in S_{22}\}$$
The list $L$ is then constructed as the join between $L_{11}$ and $L_{12}$. This means that $L$ is the sublist of $L_{11} + L_{12}$ such that for all $(x, y)$ in $L$, $x \in \mathbb{F}_p$. Accordingly, $L'$ is constructed as the join between $L_{21}$ and $L_{22}$. The authors of [DM19] argued that any two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ such that $P_1 + P_2 = P_3 = (x_3, y_3)$ with $x_3 \in \mathbb{F}_p$, satisfy a certain polynomial equation

$$f(u_1, v_1, u_2, v_2) = 0,$$

where $x_i = u_i + \alpha v_i$ for $i \in \{1, 2\}$ and where $f$ is a polynomial of constant degree obtained from Weierstraß’s equation. We give more detail on how this polynomial is constructed in section 4.1. From here, they designed an algorithm for the join which is summarised in Algorithm 2. To compute the join of two lists, say $L_1$ and $L_2$, the idea is to highlight a sublist of $L_2$, such that all points $(x_2, y_2)$ satisfy

$$\prod_{(x_1, y_1) \in L_1} f(u_1, v_1, u_2, v_2) = F(u_2, v_2) = 0. \quad (1)$$

The authors of [DM19] propose to evaluate simultaneously $F$ in all $(u_2, v_2)$ such that $(x_2, y_2) \in L_2$ in order to find this sublist. This step is described by the MultipointEvaluation procedure in Algorithm 2. Then, for each of this points...
Algorithm 2 Join\((\mathbb{a}, b), f, L_1, L_2\)

Require: An elliptic curve \(E\) over \( \mathbb{F}_{p^2} \) defined by the coefficients \(a\) and \(b\) of its Weierstrass’s equation, two lists of points \(L_1, L_2\), the polynomial \(f\) associated to \(E\).

Ensure: \(L = \{(P_0, k_1 + k_2) : P_0 = P_1 + P_2 = (x, y), (P_i, k_i) \in L_i, i \in \{1, 2\}, x \in \mathbb{F}_p\}\).

1: \(L \leftarrow \perp\)
2: \(A \leftarrow \{(u_1, v_1) : u_1 + \alpha v_1 = x_1, (x_1, y_1) \in L_1\}\)
3: \(B \leftarrow \{(u_2, v_2) : u_2 + \alpha v_2 = x_2, (x_2, y_2) \in L_2\}\)
4: \(A_i \leftarrow \{f_{u_1, v_1} \in \mathbb{F}_{q}^2[U, V] : f_{u_1, v_1}(U, V) = f(u_1, v_1, U, V)\}\)
5: \(F \leftarrow \prod_{A_i} f_{u_1, v_1}\)
6: \(E \leftarrow \text{MultipointEvaluation}(F, B)\)
7: \(B' \leftarrow \{B[i] : E[i] = 0\}\)
8: for all \(f_{u_1, v_1} \in A_i\) do
9: for all \((u_2, v_2) \in B'\) do
10: if \(f_{u_1, v_1}(u_2, v_2) = 0\) then
11: \(L \leftarrow L \cup \{(P_1 + P_2) : P_i = (x_i, y_i) \in L_i, x_i = u_i + \alpha v_i, i \in \{1, 2\}\}\)
12: return \(L\)

\(P_2\), they search all corresponding points \(P_1 \in L_1\) such that \((x, y) = P_1 + P_2\) implies \(x \in \mathbb{F}_p\). These points \(P_1\) are recovered by exhaustive search.

We also have the following result from Delaplace and May.

**Lemma 4 ([DM19] Theorem 1).** If Algorithm 2 runs in time \(T\) and uses memory \(M\) then it is possible to solve \(p^2\)-ECDLP in time mostly \(T\) using mostly \(M\) as memory.

Note that we do not actually need to store the list \(L'\). When a point \(R = Q - k_2 P\) is computed during the join procedure, we can directly check whether \(R \in L\). If so the solution can be returned immediately.

### 3.2 Improving the Join Runtime

Delaplace and May used an algorithm from Nüsken and Ziegler [NZ04] for the MultipointEvaluation procedure. Given a bivariate polynomial \(F\) of individual degrees lower than some \(d\), Nüsken and Ziegler’s algorithm evaluates this polynomial simultaneously in \(N\) points in time \(\mathcal{O}\left(\left((N + d^2)\omega_2/2^{1+\epsilon}\right)\right)\), for any fixed \(\epsilon > 0\) and where \(\omega_2\) is the linear algebra exponent of the multiplication of an \(n\)-by-\(n\) matrix by a \(n\)-by-\(n^2\) one. The authors of [DM19] showed that this step dominates the whole runtime, and for input lists \(L_1\) and \(L_2\) of respective size \(\sqrt{p}\) and \(p\), this lead to a runtime \(T = \mathcal{O}\left(p^{1.314}\right)\) – replacing \(\omega_2\) by the best known bound (cf. [LG14]) – for the whole Join procedure, and thus for their \(p^2\)-ECDLP algorithm.

Instead of Nüsken and Ziegler’s results, we propose to use Kedlaya and Umans’s algorithm whose complexity is recalled in Lemma 2. This leads to the following result.

**Lemma 5.** Let \(L_1\) and \(L_2\) be such that \(|L_1|^2 = |L_2| = p^c\), where \(c \leq 1\) is a constant. For any fixed \(\epsilon > 0\), Algorithm 2 computes the join between \(L_1\) and \(L_2\) in time \(\mathcal{O}\left(|L_2|^{1+\epsilon}\right)\).
Fig. 2: Bits repartition of elements from $S_{11}$, $S_{12} = S_{22}$, $S_{21}$ as vector of $F_{2^\log r}$, the leftmost part being the least significant bits of the numbers. The white rectangles represent a portion of zeroes.

Proof. This proof is very similar to the one given in [DM19] Theorem 2. Let $\epsilon > 0$ be a fixed constant. First we claim that steps 2 to 4 can be performed in time which is proportional to the input lists $L_1$ and $L_2$, that is in $O(|L_2|)$ operation over $F_p$. Using fast polynomial multiplication algorithms, it is possible to compute $F$ in time $\tilde{O}(|L_1|^2)$. We now need to evaluate the bivariate polynomial $F$ of individual degrees smaller than some $d = O(|L_1|)$ in $|L_2|$ points of $F_p^2$. From Lemma 2, it can be done in time $\tilde{O}((|L_1|^2 + |L_2|)^{1+\epsilon})$, since $|L_2| = p^c$ for some constant $c$. Step 7 can be performed in time $O(|L_2|)$. Finally the last part of the algorithm consists in an exhaustive search which takes time $\tilde{O}(|L_1||L_2'|)$, where $L_2'$ is the sublist of $L_2$ of the elements which pass the multipoint evaluation test.

Similar to [DM19], we rely on Schwartz-Zippel lemma and Heuristic 1 to argue that the expected value of $|B'|$ is

$$E[|B'|] \leq \frac{d|L_2|}{p} = O\left(p^{3c/2-1}\right).$$

Since $c \leq 1$, we have $3c/2 - 1 \leq c/2$ and thus $|B'| = O(|L_1|)$. It follows that this last step can be performed in time $O(|L_1|^2)$. Taking into account that $|L_1|^2 = |L_2|$ and combining all intermediary results, we get the claimed time complexity.

3.3 New Complexity Claim

With this improved join procedure, we can reduce the complexity claim of [DM19]. We have the following result.

**Theorem 1.** Using the algorithm from [KU08] as the *MultipointEvaluation* routine, for any constant $\epsilon > 0$ Algorithm 1 solves $p^2\text{-ECDLP}$ in time and memory $\tilde{O}(p^{1+\epsilon})$.

**Proof.** In [DM19] Delaplace and May chose the sets $S_{ij}$ for $i, j \in \{1, 2\}$ so that $|L_{11}| \approx |L_{21}| = O(\sqrt{p})$ and $|L_{12}| = |L_{22}| = O(p)$, (see Figure 1). More precisely,
they set

\[ S_{11} = \{ k_{11} \in \mathbb{N} : k_{11} < \lfloor \sqrt{p} \rfloor \} \]
\[ S_{21} = \{ k_{21} \in \mathbb{N} : k_{21} < p, \exists s \in \mathbb{N} : k_{21} = \lfloor \sqrt{p} \rfloor s \} \]
\[ S_{12} = S_{22} = \{ k_{12} \in \mathbb{N} : k_{12} < r, \exists s \in \mathbb{N} : k_{12} = ps \}, \]

where \( \lfloor \sqrt{p} \rfloor \) is the closest integer from \( \sqrt{p} \) (see Figure 2 to see how the bits of these elements are dispatched). As direct consequence of Lemmas 4 and 5, for any constant \( \epsilon > 0 \), Algorithm 1 solves \( p^2\text{-ECDLP} \) in time and memory \( \tilde{O}(\max(\{|L_{12}|^{1+\delta}, |L_{22}|^{1+\epsilon}\})) = \tilde{O}(p^{1+\epsilon}) \).

\[ \square \]

4 A New \( p^2\text{-ECDLP} \) Algorithm with Pre-computation

In the section, we present a new algorithm for the \( p^2\text{-ECDLP} \) problem, based on the work of [DM19].

Once again, we assume that we are given an elliptic curve \( E \) defined by its Weierstraß’s equation \( y^2 = f(x) = x^3 + ax + b \), a subgroup \( G = \langle P \rangle \) of prime order \( r = \mathcal{O}(p^2) \), and a target point \( Q \) in \( G \).

The idea is that we can pre-compute lists \( L_{ij} \), for some well chosen subset \( S_{ij} \) of \( \mathbb{N} \). It is also possible to pre-compute the join \( L \) between \( L_{11} \) and \( L_{12} \), since it does not depend on \( Q \).

The algorithm would then take as input \( L, L_{21} \) and \( L_{22} \), the point \( Q \) and the polynomial \( f \), and perform the following steps.

1. For all points \( P_{21} \in L_{21} \) replace \( P_{21} \) by \( P_{21} + Q \).
2. Compute the join \( L' \) between \( L_{21} \) and \( L_{22} \).
3. Search for a collision between \( L \) and \( L' \).

In this section, we will show that the following result holds.

**Theorem 2.** For any \( \delta > 0 \), Algorithm 5 solves \( p^2\text{-ECDLP} \) in time \( \tilde{O}(p^{2c_2+\delta}) \), using \( \tilde{O}(\max(p^{3c_1-1}, p^{2c_2+\delta})) \) memory, after a pre-computation which takes time \( \tilde{O}(p^{2c_1}) \).

The memory consumption of our algorithm is minimised when \( c_1 = \frac{3}{5} \) and \( c_2 = \frac{2}{5} \). In this case, our algorithm runs in time \( \tilde{O}(p^{4/5+\delta}) \) using \( \tilde{O}(p^{4/5+\delta}) \) memory after a pre-computation step having runtime \( \tilde{O}(p^{6/5+\delta}) \).

4.1 Pre-computation

There are basically three steps during the pre-computation: (1) compute the polynomial \( f \); (2) compute the lists \( L_{ij} \) as given in Equation 2 for some well chosen sets \( S_{ij} \), \( i, j \in \{1, 2\} \); (3) compute the join \( L \) between \( L_{11} \) and \( L_{12} \). These steps are summarised in Algorithm 3.
Algorithm 3 Precomputation\((a, b, P)\)

Require: An elliptic curve \(E\) over \(\mathbb{F}_p^2\) defined by the coefficients \((a, b)\) of its Weierstraß’s equation \(y = f(x)\), a generator \(P\) of \(G\).

Ensure: Lists \(L, L_{21}, L_{22}\), such that the \(L_{2i}\), lists are defined as in Equation 2 and \(L = \{k_1P : (x, y) : k_1 \in S_1 + S_2, x \in \mathbb{F}_p\}\) and the polynomial \(f\).

1: \(G \leftarrow \{(X_1-X_3)^2(X_2+X_3) - f(X_1) - f(X_2)^2 - 4f(X_1)f(X_2)\} F_{p^2}[X_1, X_2, X_3] \)
2: \(g_0, g_1 \leftarrow \text{Weil-Descent}(G)\) in \(F_p[U_1, U_2, V_2, U_3, V_3]\)
3: \(g_0 \leftarrow g_0(U_1, V_1, U_2, V_2, U_3, 0)\) in \(F_p[U_1, V_1, U_2, V_2, U_3]\)
4: \(g_1 \leftarrow g_1(U_1, V_1, U_2, V_2, U_3, 0)\) in \(F_p[U_1, V_1, U_2, V_2, U_3]\)
5: \(f \leftarrow \text{Res}_{U_3}(g_0, g_1)\)
6: \(L_{11} \leftarrow \{k_{11}P : k_{11} \in S_{11}\}\)
7: \(L_{12} \leftarrow \{k_{12}P : k_{12} \in S_{12}\}\)
8: \(L_{21} \leftarrow \{-k_{21}P : k_{21} \in S_{21}\}\)
9: \(L_{22} \leftarrow \{-k_{22}P : k_{22} \in S_{22}\}\)
10: \(L \leftarrow \text{Join}(a, b, L_{11}, L_{12}, f)\)
11: \(\text{return } L, L_{21}, L_{22}, f\)

More sophisticated join. If we used the procedure given in Algorithm 2 to perform this last step, then the time complexity of the join would be dominated by the exhaustive search at the end. Indeed this search takes a time which is proportional to the product of \(|L_{11}|\) and \(|L_{12}|\), where \(L_{12}\) is the list of points in \(L_{12}\) which pass the multipoint-evaluation test. From the Schwartz-Zippel Lemma, under Heuristic 1, we can expect \(|L_{12}|\) to be \(O\left(|L_{12}|^{3/2}/p\right)\), assuming that \(|L_{11}|^2 = |L_{12}|\). Now, if \(|L_{12}| \geq p\), \(|L_{12}| \geq \sqrt{p}\), and thus \(|L_{11}|\sqrt{L_{12}} \geq |L_{12}|\).

We claim that, even in this case, it is still possible to perform the join in time \(\tilde{O}\left(|L_{12}|^{3+\epsilon}\right)\), assuming that \(|L_{11}|^2 = |L_{12}| = p^{2c}\), for any constant \(c < 1\). More precisely, for any constant \(\gamma\) such that \(\frac{2^{-\gamma-1}}{2^{\gamma-1}} < c \leq \frac{2^{-\gamma-1}}{2^{\gamma-1}}\), there is an algorithm computing the join of two lists \(L_1\) and \(L_2\) such that \(|L_1|^2 = |L_2| = p^{2c}\). The Join procedure given in Algorithm 2 is a particular case of this procedure when \(\gamma = 1\).

For the sake of simplicity, we present an algorithm which does the job only for \(1/2 < c \leq 3/4\) in Algorithm 4.

The idea is to first build the list \(B'\) of all points \((u_2, v_2)\) in \(B\) such that there is a polynomial \(f_{u_2, v_2}\) in \(A_t\) such that \(f_{u_2, v_2}(u_2, v_2) = 0\). Then, for a well chosen parameter \(\kappa\), we split \(A_t\) into \(p^\kappa\) buckets \(A_t^{(i)}\), such that \(|A_t^{(i)}| = p^{\kappa}\) for all \(i\). We compute \(p^\kappa\) sublists \(B^{(i)}\) of \(B'\), such that for all \((u_2, v_2)\in B^{(i)}\), the product \(F_{u_2, v_2}\) of all \(f_{u_2, v_2}\in A_t^{(i)}\), satisfies \(F_{u_2, v_2} = 0\). We re-iterate the procedure, with the \(A_t^{(i)}\) and \(B^{(i)}\) lists, as long as required (if \(\frac{2^{-\gamma-1}}{2^{\gamma-1}} < c \leq \frac{2^{-\gamma-1}}{2^{\gamma-1}}\), then \(\gamma\) steps are to be expected).

We have following lemma.

Lemma 6. Let \(L_1\) and \(L_2\) be such that \(|L_1|^2 = |L_2| = p^{2c}\), where \(c < 1\) is a constant. For any fixed \(c > 0\), it is possible to compute the join between \(L_1\) and \(L_2\) in time \(\tilde{O}\left(|L_2|^{1+\epsilon}\right)\).
Algorithm 4 Join((a, b), f, L1, L2) for |L1| ≤ p^{3/4}

Require: An elliptic curve E over \( \mathbb{F}_p^2 \) defined by the coefficients \( a \) and \( b \) of its Weierstraß’s equation, two lists of points \( L_1, L_2 \), such that \( |L_1| = p^i |L_2| = p^{2c} \), \( 0 < c \leq 3/4 \) the polynomial \( f \) associated to \( E \).

Ensure: \( L = \{(P_0, k_1 + k_2) : P_0 = P_1 + P_2 = (x, y), (P_i, k_i) \in L_i, i \in [1, 2], x \in \mathbb{F}_p \} \).

1: \( L \leftarrow \perp \)
2: \( A \leftarrow \{(u_1, v_1) : u_1 + \alpha v_1 = x_1, (x_1, y_1) \in L_1\} \)
3: \( B \leftarrow \{(u_2, v_2) : u_2 + \alpha v_2 = x_2, (x_2, y_2) \in L_2\} \)
4: \( A_r \leftarrow \{f_{u_1, v_1} \in F_{\mathbb{F}_p^2}[U, V] : f_{u_1, v_1}(U, V) = f(u_1, v_1, U, V)\} \)
5: for all \( 1 \leq i \leq p^{c_1} \) do
6: \( A_{i+1} \leftarrow A_i(((i-1)p^{2c-1} - 1) \cdot p^{2c-1} - 1) \)
7: \( F_i = \prod_{A_{i+1}} f_{u_1, v_1} \)
8: \( F \leftarrow \prod_i F_i \)
9: \( E \leftarrow \text{MultpointEvaluation}(F, B) \)
10: \( B' \leftarrow [B[i] : E[i] = 0] \)
11: \( k_1 \leftarrow 1 - c \)
12: for all \( 1 \leq i \leq p^{c_1} \) do
13: \( E_{i+1} \leftarrow \text{MultpointEvaluation}(F, B) \)
14: \( B_{i+1} \leftarrow [B'[j] : E[i][j] = 0] \)
15: for all \( f_{u_1, v_1} \in A_{i+1} \) do
16: for all \( (u_2, v_2) \in B_{i+1} \) do
17: if \( f_{u_1, v_1}(u_2, v_2) = 0 \) then
18: \( L \leftarrow L \cup \{P_1 + P_2 : P_i = (x_i, y_i) \in L_i, x_i = u_i + \alpha v_i, i \in [1, 2]\} \)
19: return \( L \)

For simplicity we prove this lemma for the case \( 1/2 < c \leq 3/4 \). As we will see in Section 4.3, this corresponds to the case where the memory consumption of our Algorithm is the lowest \( (c = 3/5) \).

Proof. It is already clear from Lemma 5 that Steps 1 to 10 of the Join procedure described in Algorithm 4 takes time \( \tilde{O}(|L_2|^{1+\epsilon}) \). The rest of the algorithm consists mostly in computing \( p^{c_1} \) polynomial multipoint evaluations and an exhaustive search at the end.

We first estimate the cost of the multipoint evaluations. Using Kedlaya-Umans algorithm, this would be

\[
\tilde{O}\left(p^{c_1}|A_{i+1}|^2 + |B'|\right)^{1+\epsilon'},
\]

for some constant \( \epsilon' > 0 \). Furthermore for all \( i \), we have \( |A_{i+1}| = p^{2(c-k_i)} = p^{2(2c-1)} \), since \( k_1 = 1 - c \). We also have \( |B'| = p^{2c+1} \). The reader can check that \( |B'| \geq |A_{i+1}| \). It follows that the time complexity of the multipoint evaluation is \( \tilde{O}\left(p^{c_1+(3c-1)(1+\epsilon')}\right) = \tilde{O}\left(p^{2c+(3c-1)\epsilon'}\right) \). We choose \( \epsilon' \leq (3c - 1)\epsilon / 2c \) so that this complexity is \( \tilde{O}(|L_2|^{1+\epsilon}) \).

The runtime of the exhaustive search at the end is given by

\[
\mathcal{O}\left(p^{c_1}|A_{i+1}|^2 |B'|\right),
\]
Having $c < 3/4$, implies that $p^{6c-3} < |L_2|$. \hfill \Box

Using the same argument, the proof can be extended to all $\frac{2^{\gamma_i-1}-1}{2^{\gamma_i}} < c \leq \frac{2^{\gamma_i}}{2^{\gamma_i-1}}$, where $\gamma$ is a constant positive integer, by choosing a sequence of parameters $\kappa_0, \ldots, \kappa_{\gamma_i-1}$, such that $\kappa_i = i(1-c)$.

We are now ready to prove this following statement.

**Lemma 7.** For any constant $\epsilon$, it is possible to perform the whole pre-computation in $O(|L_{12}|^{1+\epsilon})$ operations over $\mathbb{F}_p$.

**Proof.** First, we claim that computing the polynomial $f$ can be done in constant time. Then, using a similar argument than the one used by [DM19] to prove Lemma 4, we show that computing the join takes more time than creating the base lists.

Let us recall that any point $(x, y)$ of $E$, $y = f(x)$. Let $G$ be the polynomial in $\mathbb{F}_p[x_1, x_2, x_3]$ such that

$$G(x_1, x_2, x_3) = ((x_1 - x_2)^2(x_1 + x_2 + x_3) - f(x_1) - f(x_2))^2 - 4f(x_1)f(x_2).$$

If $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$ are such that $P_1 + P_2 = P_3$, then $G(x_1, x_2, x_3) = 0$. Let $(1, \alpha)$ be a base of $\mathbb{F}_{p^2}$ seen as a $\mathbb{F}_p$-vector space of dimension 2. From Weil Descent, there exist polynomials $g_0$ and $g_1$ in $\mathbb{F}_p[u_1, v_1, u_2, v_2, u_3, v_3]$ such that

$$G(u_1 + \alpha v_1, u_2 + \alpha v_2, u_3 + \alpha v_3) = g_0(u_1, v_1, u_2, v_2, u_3, v_3) + \alpha g_1(u_1, \ldots, v_3).$$

$G$ is a polynomial in 3 variables of constant individual degree (i.e. $\deg_{x_i}(G) = \deg_{x_j}(G) = 6$ and $\deg_{x_k}(G) = 2$), as such, computing $g_0$ and $g_1$ can be done in constant time, and the individual degrees of $g_0$ and $g_1$ are constants. As in [DM19], we consider polynomials

$$g'_i(u_1, v_1, u_2, v_2, u_3) = g_i(u_1, v_1, u_2, v_2, u_3, 0) \quad \text{for } i \in \{0, 1\}.$$

and we compute the resultant $f$ of $g'_0$ and $g'_1$ in $U_3$. As both $g_i$ polynomials consist of a constant number of variables and have constant individual degrees, this can be performed in constant time.

As already argued in [DM19], building the lists $L_{12}, L_{12}, L_{21}, L_{22}$ can be done in $O(\max_i(|L_i|))$ operations over $\mathbb{F}_p$. In order to minimise the runtime of the join routine, we choose $|L_{11}|^2 = |L_{12}|$ and $|L_{21}|^2 = |L_{22}|$. We would also like to
Algorithm 5 DM19-ECDLP((a, b), Q, L, L_{21}, L_{22}, f)

Require: An elliptic curve \( E \) over \( \mathbb{F}_p \) defined by the coefficients \((a, b)\) of its Weierstrass's equation, a target point \( Q \), the lists \( L, L_{21} \) and \( L_{22} \) the polynomial \( f \) associated to \( E \).

Ensure: \( k \in \mathbb{Z}_r \) such that \( kP = Q \) or \( \bot \).

1: \( L_{21} \leftarrow L_{21} + \{Q\} \)
2: \( L' \leftarrow \text{Join}(a, b, L_{21}, L_{22}, f) \)
3: for all \( Q - k_2P \in L' \) do
4: \( \text{if } Q - k_2P = k_1P \in L \) then
5: \( \text{return } k_1 + k_2 \)

compute the join between \( L_{21} \) and \( L_{22} \) faster than the one between \( L_{11} \) and \( L_{12} \), as this join would be part of the actual algorithm. So it would be better if the lists \( L_{21} \) and \( L_{22} \) are smaller than \( L_{11} \) and \( L_{12} \). It follows that \( L_{12} \) is the largest list to be computed, and thus the runtime of this step is \( \tilde{O}(|L_{12}|) \) elementary operations over \( \mathbb{F}_p \).

Finally we need to compute the join between \( L_{11} \) and \( L_{12} \). From Lemma 6, this can be performed in \( \tilde{O}(|L_{12}|^{1+\epsilon}) \) for any chosen constant \( \epsilon > 0 \). \( \square \)

4.2 The Main Algorithm

Given \( Q \) and the precomputed lists \( L, L_{21}, L_{22} \) we want to recover \( k \) such that \( Q = kP \). The method is summarised in Algorithm 5.

Lemma 8. Given \( L, L_{21}, L_{22} \) as described in the previous section such that \( |L_{21}| < p \), for any fixed constant \( \epsilon > 0 \), Algorithm 5 runs in time \( \tilde{O}(|L_{22}|) \) and uses \( \tilde{O}(\max(|L|, |L_{22}|^{1+\epsilon})) \) elements of \( \mathbb{F}_p \) as storage.

Proof. The first step of the algorithm consists in replacing all points \( P_3 \) of \( L_{21} \) by \( Q + P_3 \), this can be done in \( O(|L_{21}|) \) elementary operations over \( \mathbb{F}_p \). For now on, we consider that \( L_{21} \) is the list of all \( Q - k_{21}P \).

The second step consists in building the join \( L' \) between \( L_{21} \) and \( L_{22} \). For any constant \( \epsilon > 0 \), using Kedlaya-Umans to perform the multi-evaluation step, it is possible to compute \( L' \) in time \( \tilde{O}(|L_{22}|^{1+\epsilon}) \), provided that \( |L_{22}| = |L_{21}|^2 \).

Finally the last step consists in searching for a collision between the lists \( L \) and \( L' \). We can assume that \( L \) is sorted according to the \( x \)-coordinate of its points (sorting \( L \) could have been done during the pre-computation step in time \( O(|L|) \) under Heuristic 1). Under Heuristic 1, it is possible to check if a point \( R = (x, y) \), with \( x \in \mathbb{F}_p \) belongs to \( L \) in constant time. As such this whole step can be performed in time \( \tilde{O}(|L'|) \).

This gives us a total time complexity of \( \tilde{O}(|L_{21}| + |L_{22}|^{1+\epsilon} + |L'|) \). We have \( |L_{21}|^2 = |L_{22}| \) as such \( |L_{21}| < |L_{22}| \). Furthermore, let us denote by \( L'_{22} \) the sublists of \( L_{22} \) of all points \((x_2, y_2)\) with \( x_2 = u_0 + \alpha v_2 \) such that there is \((x_1, y_1) \in L_{21} \) with \( x_2 = u_2 + \alpha v_2 \) and \( f(u_1, v_1, u_2, v_2) = 0 \). Under Heuristic 1,
we can show that the expected size of $L'$ is

$$\mathbb{E}[|L'|] = O \left( \frac{|L_{21}| |L_{22}|}{p} \right).$$

This comes from the fact that if the $x$-coordinates of the points of $L_{21} + L_{22}$ are uniformly distributed, then a point $(x, y) \in L_{21} + L_{22}$ will satisfy $x \in \mathbb{F}_p$ with probability $p^{-1}$. Having $|L_{21}| < p$ ensures that $\mathbb{E}[|L'|] < |L_{22}|$. Thus the whole time complexity of this algorithm is

$$T = \tilde{O} \left( |L_{22}|^{1+\epsilon} \right).$$

We now prove the claimed memory complexity. This algorithm requires to store the input lists $L, L_{21}$ and $L_{22}$, as well as the list $L'$. Some additional lists have to be stored during the join procedure, but their size is bounded by $\tilde{O} \left( |L_{22}|^{1+\epsilon} \right)$. We have already argued that $\max(|L'|, |L_{21}|) \leq |L_{22}|$. It follows that the memory required by this algorithm is

$$\tilde{O} \left( \max(|L|, |L_{22}|^{1+\epsilon}) \right),$$

which concludes the proof.

\[\square\]

4.3 Complexity Analysis

Choice of the $S_{ij}$ sets. Now we discuss how to define the sets $S_{ij}$. Let $\frac{1}{2} \leq \lambda \leq 1$ be a parameter. For simplicity, we assume that $p^\lambda$ is an integer. If not, replacing $p^\lambda$ by its rounding to the closest integer, the argument still holds. We consider the following subsets of $\mathbb{N}$.

$$S_1 = \{k_1 < r : k_1 = s_1 p + s_0, 0 \leq s_0 < p^\lambda \}$$

$$S_2 = \{k_2 < r : k_2 = s_1 p + s_0, 0 \leq s_0 < p \text{ and } p^\lambda | s_0 \}$$

The shape of the elements from the $S_i$ sets is given in Figure 3a. The sets $S_{ij}$ are built so that $S_{i1} \cap S_{i2} = \{0\}$, $S_{i1} + S_{i2} = S_i$ and $|S_{i1}|^2 = |S_{i2}|$ for $i \in \{1, 2\}$. More precisely let $c_1 = \frac{1}{3}(\lambda + 1)$ and $c_2 = \frac{1}{3}(2 - \lambda)$. It is possible to check that the following holds:

$$\frac{1}{2} \leq c_1 \leq \lambda \leq 1 \leq \lambda + c_2 \leq \frac{4}{3},$$

since, $\frac{1}{2} \leq \lambda \leq 1$. We construct the sets $S_{ij}$ as follows.

$$S_{11} = \{k_{11} \in S_1 : k_{11} < p^{c_1} \}$$

$$S_{12} = \{k_{12} \in S_1 : k_{12} = s_1 p + s_0, p^{c_1} | s_0 \}$$

$$S_{21} = \{k_{21} \in S_2 : k_{21} < p^{\lambda+ c_2} \}$$

$$S_{22} = \{k_{22} \in S_2 : k_{22} = s p, p^{\lambda+ c_2 - 1} | s \}$$

The shape of the elements from the $S_{ij}$ sets is given in Figure 3b.
Lemma 9. The sets $S_i$ and $S_{ij}$ satisfies $S_i \cap S_{ij} = \{0\}$, $S_i + S_{ij} = S_i$, and $|S_i|^2 = O(|S_{ij}|)$, for $i \in \{1, 2\}$.

Proof. Let us first prove the relations between the sizes. By construction $|S_{11}| = p^{c_1}k_{12} < r$ and $0 \leq s_0 < p^\lambda$, $|S_{12}| = \frac{c_1}{p} \cdot p^{\lambda - c_1} = O(p^{1+\lambda - c_1})$. Recall that $c_1 = \frac{1}{3}(1 + c_1)$, it follows that

$$|S_{12}| = O(p^{2c_1}).$$

We also have $|S_{21}| = \frac{p^{c_2}}{p^{\lambda + c_2}} = p^{c_2}$, and $|S_{22}| = \frac{c_2}{p} \cdot \frac{1}{p^{\lambda + c_2}} = O(p^{2-\lambda - c_2})$. Recall that $c_2 = \frac{1}{3}(2 - \lambda)$, it follows that

$$|S_{22}| = O(p^{2c_2}).$$

We prove that the intersection is reduced to $\{0\}$. It is clear that 0 belongs to all sets. Let now be $k_{12} = s_1p + s_0p^{c_1}$ an element of $S_{12}$. We show that if $k_{12} \in S_{11}$, then $k_{12} = 0$. Indeed $k_{12} \in S_{11}$ means that $k_{12} < p^{c_1}$ this implies that $s_0 = s_1 = 0$. The same argument can be used to show that $S_{21} \cap S_{22} = \{0\}$.
Finally we need to show that $S_{11} + S_{12} = S_i$. For any $k_1 \in S_{11}$, $k_1 = s_1p + s_0$, where $s_0 < p^\lambda$. Furthermore $s_0$ can be expressed as $s_0 = s_01p^{c_1} + s_{00}$, for $0 \leq s_{00} < p^{c_1}$. Setting $k_1 = s_1p^{c_1} + s_{01}p^{c_2} \in S_{11}$ and $k_2 = s_00 \in S_{12}$, we have $k_1 = k_{11} + k_{12} \in S_{11} + S_{12}$. Conversely, if $k_1 = k_{11} + k_{12} \in S_{11} + S_{12}$, we have $k_1 = s_1p + s_0 + k_{11}$, such that $s_0 = ap^{c_1} < p^\lambda$ and $k_{11} < p^{c_1}$. We have $s_0' = s_0 + k_{11} < p^\lambda$. As such $k_1 \in S_{11}$. A similar argument can be used to show that $S_{21} + S_{22} = S_2$.

We also have the following result, similar to [DM19] Lemma 1.

**Lemma 10.** There are at least $\frac{c}{p} + 1$ representations of $k$ as the sum $k_1 + k_2$ with $(k_1, k_2) \in S_1 \times S_2$.

**Proof.** Let $k' \in \mathbb{N}$ such that $k' \mod r = k$, $0 \leq k' < r$. Let $0 \leq k_0' < p$ and $a'$ be the unique integers such that $k' = a'p + k_0'$, and let $0 \leq c' < p^\lambda$ and $b'$ be the unique integers such that $k_0' = b'p^\lambda + c'$. Let $S'_1$ be the subset of $S_1$ such that $S'_1 = \{k'_1 = c' + s_1p\}$. We show that for any $k'_1 \in S'_1$ such that $k'_1 < k'$, there exist a $k'_2 \in S_2$ such that $k = k'_1 + k'_2 \mod r = k' \mod r$.

For any $k'_1 \in S'_1$, $k'_1 \leq k'$, there is some $0 \leq s_1 \leq a' < r/p$ such that $k'_1 = s_1p + c'$. Let $0 \leq s_2 \leq a'$ be such that $s_2 + s_1 = a'$, and let $k'_2 = b'p^\lambda + s_2p$. It is clear that $k'_2 \in S_2$, and $k'_1 + k'_2 = c' + b'p^\lambda + (s_1 + s_2)p = k'$.

Similarly, if we consider $k'' \in \mathbb{N}$ such that $k'' = k' + r$, we can find unique $a'', b'', c''$ such that $k'' = a''p + b''p^\lambda + c''$ such that $0 \leq c'' < p^\lambda$ and $0 \leq b''p^\lambda + c'' < p$. We consider the sublist $S''_1$ of $S_1$ of all $k''_1$ such that $k''_1 = c'' + s_1p$. With the same argument, we can show that for all $k''_1 \in S''_1$ such that $k''_1 > k'$, there is a $k''_2$ such that $k''_2 = b''p^\lambda + s_2p$, with $s_2 + s_1 = a''$ such that $k''_1 + k''_2 = k''$. Furthermore, since $k''_1 > k'$, $k''_2 = k'' + r - k''_1 < r$, which proves that $k''_2$ is indeed in $S_2$.

It follows that the number of representations $(k_1, k_2) \in S_1 \times S_2$ of $k$ is at least equal to some $N = N_1 + N_2$, where $N_1$ is the number of elements $k'_1$ of $S'_1$ such that $k'_1 < k'$ and $N_2$ is the number of elements $k''_1$ of $S''_1$ such that $k''_1 > k'$. We have $N_1 = a' + 1$ and $N_2 \approx r/p - a'$ it follows that $N \approx r/p + 1$, which concludes the proof. \(\square\)

**Complexity.** Using the results above, we are ready to prove Theorem 2.

**Proof (of Theorem 2).** For any $\delta > 0$, we set $\epsilon = \delta/(2c_2)$. With our definition of the $S_{ij}$ sets, we have

$$|L_{11}| = p^{c_1}, \quad |L_{12}| = O\left(p^{2c_1}\right),$$

$$|L_{21}| = p^{c_2}, \quad |L_{22}| = O\left(p^{2c_2}\right).$$

Furthermore, under Heuristic 1, we have $|L| = |L_{11}| |L_{12}|/p = p^{3c_1-1}$. Combining this with the results of Lemma 8 and Lemma 7, we obtain that Algorithm 5 runs in time $\tilde{O}\left(p^{2^{(c_1+\epsilon)}}\right) = \tilde{O}\left(p^{2^{c_2+\delta}}\right)$, using $\tilde{O}\left(\max\left(p^{3c_1-1}, p^{2^{c_2+\delta}}\right)\right)$, after a precomputation step which takes time $\tilde{O}\left(p^{2^{c_1+\delta}}\right)$. 

18
Now, if we set the parameter $\lambda$ defined in the previous subsection to $\lambda = \frac{4}{5}$ we have $c_1 = \frac{3}{5}$ and $c_2 = \frac{2}{5}$, this gives us
\[
3c_1 - 1 = \lambda = \frac{4}{5} = 2c_2.
\]
In this case $|L| = |L_2|$. For larger values of $\lambda$ the size of $L$ increases, and for smaller values of $\lambda$ the size of $L_2$ increases, which makes $\lambda = \frac{4}{5}$ the value for which the memory consumption of this algorithm is minimised.

\[
\square
\]

5 Extending the Algorithm to Other Fields

Here we investigate an extension of this algorithm to fields $\mathbb{F}_{p^\ell}$, where $\ell \geq 2$ is a constant.

Let $(1, \alpha, \ldots, \alpha^{\ell-1})$ be a basis of $\mathbb{F}_{p^\ell}$ seen as an $\mathbb{F}_p$-vector space. We consider an elliptic curve $E$ over $\mathbb{F}_{p^\ell}$ defined by its Weierstraß's equation: $y = f(x) = x^3 + ax + b$, a subgroup $\mathcal{G} = \langle P \rangle$ of prime order $r = \mathcal{O}(p^2)$ and a target point $Q \in \mathcal{G}$.

Once again, the idea is to split the problem into four lists $L_{11}, L_{12}, L_{21}$ and $L_{22}$ such that
\[
L_{1j} = \{k_{1j}P : k_{1j} \in \mathcal{S}_{1j} \}
\]
\[
L_{2j} = \{-k_{2j}P : k_{2j} \in \mathcal{S}_{2j} \}.
\]
For well chosen subsets $\mathcal{S}_{ij}$ of $\mathbb{N}$. These lists, which do not depend on $Q$ can be pre-computed. We can also pre-compute the join $L$ between $L_{11}$ and $L_{12}$ such that all points $(x, y) \in L$ satisfy $x = u_0 + \alpha u_1 + \cdots + \alpha^{\ell-2} u_{\ell-2}$. The main algorithm would then consist in replacing $L_{21}$ by $L_{21} + \{Q\}$, compute the join $L'$ between $L_{21}$ and $L_{22}$ and search for a collision between $L$ and $L'$.

The pre-computation is given in Algorithm 6 and the main procedure in Algorithm 7.

In this section, we prove the following theorem.

**Theorem 3.** For any $\delta > 0$, Algorithm 7 solves ECDLP, with base field $\mathbb{F}_{p^\ell}$ in time $\tilde{O}(p^{\ell c_2 + \delta})$, using $\tilde{O}(\max(p^{(\ell+1)c_1-1}, p^{\ell c_2 + \delta}))$ memory, after a pre-computation which takes time $\tilde{O}(p^{2c_1})$.

The memory consumption of our algorithm is minimised when $c_1 = \frac{\ell+1}{2\ell+1}$ and $c_2 = \frac{\ell}{2\ell+1}$. In this case, our algorithm runs in time $\tilde{O}(p^{\ell c_2 + \delta})$ using $\tilde{O}(p^{\ell c_2 + \delta})$ memory after a pre-computation step having runtime $\tilde{O}(p^{\ell c_2 + \delta})$.

The idea is quite similar to the one that has been developed in the previous section. We simply give more details below.
Algorithm 6 Precomputation(\((a, b), P\))

Require: An elliptic curve \(E\) over \(\mathbb{F}_{p^e}\) defined by the coefficients \((a, b)\) of its Weierstrass’s equation \(y = f(x)\), a generator \(P\) of \(G\).

Ensure: The lists \(L, L_{21}, L_{22}\), and the polynomial \(f\).
1: \(G \leftarrow [(X_1 - X_2)^2(X_1 + X_2 + X_3) - f(X_1) - f(X_2))^2 - 4f(X_1) f(X_2)] \in \mathbb{F}_{p^e}[X_1, X_2, X_3]
2: \(g_0, g_1, \ldots, g_{\ell - 1} \leftarrow \text{Weil-Descent}(G)\) in \(\mathbb{F}_p[U_1, 0, \ldots, U_{1, \ell - 1}, U_2, 0, \ldots, U_{2, \ell - 1}, U_3, 0, \ldots, U_{3, \ell - 1}]\)
3: for all \(0 \leq i < \ell\) do
   4: \(g_i \leftarrow g_i(U_1, 0, \ldots, U_3, \ell - 2, 0)\) in \(\mathbb{F}_p[U_1, 0, \ldots, U_{3, \ell - 2}]\)
5: \(f \leftarrow \text{Elimination}(g_0, \ldots, g_{\ell - 1}, U_3, 0, \ldots, U_{3, \ell - 2})\)
6: \(L_{11} \leftarrow \{k_{11} P : k_{11} \in S_{11}\}\)
7: \(L_{12} \leftarrow \{k_{12} P : k_{12} \in S_{12}\}\)
8: \(L_{21} \leftarrow \{-k_{21} P : k_{21} \in S_{21}\}\)
9: \(L_{22} \leftarrow \{-k_{22} P : k_{22} \in S_{22}\}\)
10: \(L \leftarrow \text{Join}((a, b), L_{11}, L_{12}, f)\)
11: return \(L, L_{21}, L_{22}, f\)

Algorithm 7 DM19-ECDLP(\((a, b), Q, L, L_{21}, L_{22}, f\))

Require: An elliptic curve \(E\) over \(\mathbb{F}_{p^e}\) defined by the coefficients \((a, b)\) of its Weierstrass’s equation, a target point \(Q\), the lists \(L, L_{21}\) and \(L_{22}\) the polynomial \(f\) associated to \(E\).

Ensure: \(k \in \mathbb{Z}_r\) such that \(k P = Q\) or \(\perp\).
1: \(L_{21} \leftarrow L_{21} + \{Q\}\)
2: \(L' \leftarrow \text{Join}((a, b), L_{21}, L_{22}, L, f)\)
3: for all \(Q - k P \in L'\) do
   4: if \(Q - k P = k_1 P \in L\) then
      5: return \(k_1 + k_2\)

5.1 Analysis of the Algorithm

Computing the join. We generalise the result of lemma 6 to the case of an elliptic curve defined over \(\mathbb{F}_{p^e}\).

Lemma 11. Let \(L_1\) and \(L_2\) be two lists of points of \(E(\mathbb{F}_{p^e})\), and let \(f \in \mathbb{F}_p[U_1, 0, \ldots, U_{1, \ell - 1}, U_2, 0, \ldots, U_{2, \ell - 1}]\) be a constant degree polynomial. It is possible to compute the list \(L\) such that:

\[
L = \{P_1 + P_2, P_1 = (x_1, y_1) \in L_1, P_2 = (x_2, y_2) \in L_2 : f(u_{1,0}, \ldots, u_{2,\ell - 1}) = 0\},
\]

in time and space \(\tilde{O}(|L_2|^{1+\varepsilon})\) provided that \(|L_1|^\varepsilon = |L_2| = p^{\varepsilon c}\), where \(0 < c < 1\).

Once again, for simplicity we give the proof for \(0 < c \leq 1/2\) and for \(1/2 < c \leq 3/4\). We claim that the proof can be extended for any \(c\) such that \(\frac{2^{1-c}-1}{2^{1-c}+1} < c \leq \frac{2^{1-c}+1}{2^{1-c}+1}\), where \(\gamma\) is any chosen constant.

Proof. Let \(0 < c \leq 3/4\).
The first step of the join computation is to build the lists $A_i$ and $B$, such that $A_i$ is the list of polynomials $f_{x_i}$ in $F_p[u_2,0,\ldots,u_2,\ell-1]$ with

$$f_{x_i}(u_2,0,\ldots,u_2,\ell-1) = f(u_1,0,\ldots,u_1,\ell-1,u_2,0,\ldots,u_2,\ell-1),$$

for all $(x_1,y_1) \in L_1$, and

$$B = \{(u_2,0,\ldots,u_2,\ell-1) : x_2 = u_2,0 + \cdots + \alpha^{\ell-1}u_2,\ell-1, (x_2,y_2) \in L_2\}.$$ 

We claim that this can be performed in time and memory $\mathcal{O}(\max(|L_1|,|L_2|)) = \mathcal{O}(|L_2|)$.

The second step consists in computing the polynomial $F = \prod A_i f_{x_i}$. Using fast polynomial this can be done in time and memory $\tilde{O}(|L_1|^\ell)$.

In the third step we need to compute $E = \text{MultipointEvaluation}(F,B)$. Using the Kedlaya-Umans Algorithm, for any $\epsilon > 0$ this can be done in time and memory $\tilde{O}\left(\left(|L_1|^\ell + |L_2|\right)^{1+\epsilon}\right) = \tilde{O}(|L_2|^{1+\epsilon})$. Afterward, we compute $B'$ the sublist of $B$ of all points $(u_2,0,\ldots,u_2,\ell-1)$ such that $F(u_2,0,\ldots,u_2,\ell-1) = 0$. This can be done in time and memory $\tilde{O}(|L_2|)$.

Now, if $0 < c \leq 1/2$, the fourth and last step consists in finding all pairs $(f_{x_1},(u_2,i))_{i=0}^{\ell-1}$ such that $f_{x_1}(u_2,0,\ldots,u_2,\ell-1) = 0$. This is performed by an exhaustive search which takes time $\tilde{O}(|L_1||B'|)$. We can estimate the size of $B'$ using Schwartz-Zippel lemma. We have

$$|B'| = \mathcal{O}\left(|L_1||L_2|/p\right) = \mathcal{O}\left(p^{(\ell+1)c-1}\right).$$

It follows that $|L_1||B'| = \mathcal{O}\left(p^{(\ell+2)c-1}\right)$. And $(\ell+2)c - 1 \leq \ell c$ when $c \leq 1/2$. As such $|L_1||B'|$

If $1/2 < c \leq 3/4$, we use a similar trick than in Section 4.1. We dispatch $A_i$ into $p^{\kappa_1}$ buckets $A_i^{(i)}$ such that $|A_i^{(i)}| = p^{c-\kappa_1}$, where $\kappa_1 = 1 - c$. We compute all the $E^{(i)}$ lists such that $E^{(i)} = \text{MultipointEvaluation}(F_i,B')$ where $F_i = \prod A_i^{(i)} f_{x_i}$. Then we compute the list $B^{(i)} = \{B'[j] : E^{(i)} = 0\}$. For any constant $\epsilon'$ this step can be performed in time and memory $\mathcal{O}\left(p^{\kappa_1}\left(|A_i^{(i)}|^\ell + |B'|\right)^{1+\epsilon'}\right)$.

Furthermore, we have $|A_i^{(i)}|^\ell = p^{(c-\kappa_1)\ell} = p^{2c-1}\ell, |B'| = p^{(\ell+1)c-1}$. It follows that $|B'| > |A_i^{(i)}|^\ell$ and thus the complexity of this step is

$$\tilde{O}\left(p^{\kappa_1}|B'|^{1+\epsilon'}\right) = \tilde{O}\left(p^{(2c-1)\ell c-1+\epsilon'}\right) = \tilde{O}\left(|L_2|^{1+\epsilon}\right),$$

for $\epsilon = ((\ell + 1)c - 1)\epsilon'/(\ell c)$.

The last step consist in finding all pairs $(f_{x_1},(u_2,i))_{i=0}^{\ell-1}$ such that $f_{x_1}(u_2,0,\ldots,u_2,\ell-1) = 0$. This is performed by an exhaustive search which takes time $\tilde{O}\left(p^{\kappa_1}|A_i^{(i)}||B^{(i)}|\right) = \tilde{O}\left(|L_1||B^{(i)}|\right)$. We estimate $|B^{(i)}|$ with Schwartz-Zippel Lemma. $|B^{(i)}| = \mathcal{O}\left(p^{(\ell+3)c-3}\right)$. It follows that

$$|L_1||B^{(i)}| = \mathcal{O}\left(p^{(\ell+4)c-3}\right)$$

21
and \((\ell+4)c - 3 \leq \ell c\) when \(c \leq 3/4\). As such \(|L_1|B^{(i)} = \mathcal{O}(|L_2|)\), which conclude the proof for \(c \leq 3/4\).

This proof can be extended to \(\frac{1}{2^{\gamma + 1}} c - 3 \leq \frac{1}{2^{\gamma - 1}}\), where \(\gamma\) is any chosen constant, by repeating the same argument \(\gamma\) times, and choosing a sequence of parameter \(\kappa_0, \ldots, \kappa_{\gamma - 1}\) such that \(\kappa_i = i(1 - c)\).

\begin{proof}

Pre-computation step. We show the following result, regarding the complexity of the pre-computation step.

**Lemma 12.** For any constant \(\epsilon > 0\), the pre-computation step given in Algorithm 6 can be performed in \(\mathcal{O}(|L_{12}|^{1+\epsilon})\) operations over \(\mathbb{F}_p\), for a “good choice” of the \(S_{ij}\) sets.

**Proof.** Similar to the proof of Lemma 7, we need three steps in this proof. The first step consists in showing that computing the polynomial \(f\) requires a constant number of operations over \(\mathbb{F}_p\). Then we argue that building the lists \(L_{11}, L_{12}, L_{21}\) and \(L_{22}\) requires a time that is proportional to the size of the largest one, that is, following the same argument than in the proof of Lemma 7, \(\mathcal{O}(|L_{12}|)\) for a good choice of the sets \(S_{ij}\). The last step, finally, consists in showing that the join of \(L_{11}\) and \(L_{12}\) can be computed in time \(\tilde{\mathcal{O}}(|L_{12}|^{1+\epsilon})\). Assuming that \(|L_{11}|^\ell = |L_{12}| < p^\ell\) (this can be enforced by the choice of the \(S_{ij}\) sets), this is a direct consequence of Lemma 11.

In the end, all we need to prove is that \(f\) can be computed in constant time. The argument is quite similar to the one given in the proof of Lemma 7. Since \(G\) is a polynomial of constant individual degree and \(\ell\) is a constant as well, the polynomials \(g_0, \ldots, g_{\ell - 1}\) would be polynomial of constant degree individual degree as well, and would have a constant number of variables. Furthermore their computation can be done in constant time. Computing \(f\) would require to compute the elimination ideal of \(\langle g_0, \ldots, g_{\ell - 1}\rangle\) in the variables \(\{U_3, 0, \ldots, U_3, \ell - 2\}\). As long as the number of variables of each polynomials and the individual degree of these polynomial are all constants, this can be performed in constant time as well. \(\square\)

Main algorithm. We show the following result, regarding the complexity of the main algorithm.

**Lemma 13.** For any constant \(\epsilon > 0\), Algorithm 7 solves ECDLP for an elliptic curve defined over a finite field \(\mathbb{F}_p\), in time \(\tilde{\mathcal{O}}(|L_{21}|^{1+\epsilon})\) and memory \(\tilde{\mathcal{O}}(\max(|L|, |L_{21}|^{1+\epsilon}))\).

**Proof.** Replacing the points \(P_1\) of \(L_{21}\) by \(P_1 + Q\) can be done in \(\mathcal{O}(|L_{21}|)\) operations over \(\mathbb{F}_p\). We have already argued that if \(|L_{21}|^\ell = |L_{22}| < p^\ell\), for any constant \(\epsilon > 0\), the join between \(L_{21}\) and \(L_{22}\) can be performed in time \(\tilde{\mathcal{O}}(|L_{22}|^{1+\epsilon})\). From here, the proof is trivial. \(\square\)
Combining Lemma 13 and Lemma 12, we obtain that for any constant $c$ the time complexity of the main algorithm is $O(\tilde{\omega}^2)$. Let $\lambda \leq \ell - 1$ be a parameter. We assume that $p^\lambda$ is an integer. If not the following stills holds by replacing $p^\lambda$ by its rounding to the closest integer. Let $S_1$ and $S_2$ be the following subsets of $\mathbb{N}$.

\[ S_1 = \{ k_1 < r : k_1 = s_1 p + s_0, 0 \leq s_0 < p^\lambda \} \]  
\[ S_2 = \{ k_2 < r : k_2 = s_1 p + s_0, 0 \leq s_0 < p^\lambda | s_0 \}. \]

Similar to the choice that has been done in Section 4.3, the $S_{ij}$ sets are then built so that $S_{11} \cap S_{12} = \emptyset$, $S_{11} + S_{12} = S_i$ and $|S_{11}| = |S_{12}|$. For instance, let $c_1 = \frac{\lambda + 1}{\ell - 1}$ and let $c_2 = \frac{\lambda}{\ell - 1}$. We have

\[ 0 < c_1 \leq \lambda \leq \ell - 1 \leq \ell c_2 + \lambda < \ell. \]

Following the same kind of argument than the ones given in the proof of Lemma 9 the reader can check that if we choose the $S_{ij}$ sets to be

\[ S_{11} = \{ k_{11} \in S_1 : k_{11} < p^{c_1} \} \]  
\[ S_{12} = \{ k_{12} \in S_1 : k_{12} = s_1 p^{\ell - 1} + s_0, p^{c_1} | s_0 \} \]  
\[ S_{21} = \{ k_{21} \in S_2 : k_{21} = sp^{-1}, p^{(c_2-1)+\lambda+1} | s \} \]  
\[ S_{22} = \{ k_{22} \in S_2 : k_{22} = s_1 p^{\ell - 1} + s_0, s_1 < p^{(c_2-1)+\lambda+1}, p^\lambda | s_0 \}, \]

these conditions are satisfied. The shape of the elements of the $S_{ij}$ sets defined above is illustrated in Figure 4b.

**Lemma 14.** There are at least $\frac{\ell}{p} + 1$ representations of $k$ as the sum $k_1 + k_2$ with $(k_1, k_2) \in S_1 \times S_2$.

The proof of this lemma is very similar to the one of Lemma 10, as such we do not detail it here.

**Proof of Theorem 3.** Combining all these results, we are ready to prove Theorem 3.

*Proof.* Let the $S_{ij}$ sets be defined as above. This gives us $|L_{11}| = p^{c_1}$, $|L_{12}| = O(p^{c_1})$, $|L_{21}| = O(p^{c_2})$ and $|L_{22}| = p^{c_2}$. It follows that $|L| = p^{(\ell + 1)c_1 - 1}$. Combining Lemma 13 and Lemma 12, we obtain that for any constant $\delta > 0$, the time complexity of the main algorithm is $O(p^{c_2 + \delta})$ and the memory required is $O((\max(p^{c_2+\delta}, p^{(\ell + 1)c_1 - 1}))$ after a pre-computation which requires $O(p^{c_1 + \delta})$.

The memory consumption is reduced when $\ell c_2 = (\ell + 1)c_1 - 1$. Our choice of $c_1 = \frac{\ell + 1}{2\ell + 1}$ and $c_2 = \frac{\ell}{2\ell + 2}$, is derived from this equation and from the fact that $c_1 + c_2 = 1$.

### 5.2 Discussion

We now address some open questions and remarks regarding these algorithm and their possible improvements.
(a) Bits repartition of elements from $S_1$ and $S_2$, as vector of $\mathbb{F}_2^{\log r}$, the leftmost part being the least significant bits of the numbers. The white rectangles represent a portion of zeroes.

(b) Bits repartition of elements from $S_{11}$, $S_{12}$, $S_{21}$ and $S_{22}$ as vector of $\mathbb{F}_2^{\log r}$, the leftmost part being the least significant bits of the numbers. The white rectangles represent a portion of zeroes.

Fig. 4: Bits repartition of the elements of $S_i$ and $S_{ij}$ for $i \in \{1, 2\}$, $j \in \{1, 2\}$.

Reducing the memory consumption? As already mentioned, the main bottleneck of our algorithm is the large memory required to store $L$ as soon as the parameter $c_1$ gets larger than $\ell + \frac{1}{2\ell+1}$. Note that this gets worse when $\ell$ increases. If we could find a way to reduce the storage of $L$, we should be able to extend the results given in Theorem 3 to a larger set of parameters, and get closer to Bernstein and Lange $p^{\ell/3}$ complexity, or possibly reach it for $\ell > 2$. However doing this without increasing the time complexity of the algorithm nor the pre-computation does not appear to be an easy task.

Extending the algorithm to other fields? Another interesting open question is whether our algorithm can be extended to other fields. Here, we focused on fields $\mathbb{F}_p^\ell$ for (small) constant $\ell$ and large (prime) $p$. Another cryptographically relevant case is the one where $\ell$ is a large prime and $p$ a small prime (e.g. $p = 2$). It would be possible to extend the algorithms described here to this case with some changes to make it fit the model. However, similar to what is shown in
Lemma 12 to compute the join, we would need to consider polynomials $f_1, \ldots, f_\ell$, in $O(\ell)$ variables and compute a Gröbner basis to discard the dependency in the last variable $x_3$. This would already cost more than we can afford during the pre-computation step. It could also be nice to extend the method to a field $\mathbb{F}_p$ where $p$ is a prime.

References


