Discrete Logarithms

Aurore Guillevic, François Morain

Chapter 9

Discrete Logarithms [1]

Aurore Guillevic
INRIA-Saclay and École Polytechnique/LIX
François Morain
École Polytechnique/LIX and CNRS and INRIA-Saclay

Contents

The Discrete Logarithm Problem (DLP) is one of the most used mathematical problems in asymmetric cryptography design, the other one being the integer factorization. It is intrinsically related to the Diffie-Hellman problem (DHP). DLP can be stated in various

[1] This chapter is the 9-th chapter of the book *Guide to Pairing-Based Cryptography*, edited by Nadia El Mrabet and Marc Joye, and published by CRC Press, ISBN 9781498729505. This document is the authors' version archived on HAL (`hal.inria.fr`) on December 20, 2016. The publisher's official webpage of the book is: `https://www.crcpress.com/Guide-to-Pairing-Based-Cryptography/El-Mrabet-Joye/p/book/9781498729505`

groups. It must be hard in well-chosen groups, so that secure-enough cryptosystems can be built. In this chapter, we present the DLP, the various cryptographic problems based on it, the commonly used groups, and the major algorithms available at the moment to compute discrete logarithms in such groups. We also list the groups that must be avoided for security reasons.

Our computational model will be that of classical computers. It is to be noted that in the quantum model, DLP can be solved in polynomial time for cyclic groups [108].

9.1 Setting and First Properties

9.1.1 General Setting

Let G be a finite cyclic group of order N generated by g. The group law on G is defined multiplicatively and noted $\circ$, the neutral element is noted 1_G, and the inverse of an element a will be noted $1/a = a^{-1}$. The *discrete logarithm problem* is the following:

DLP: *given $h \in G$, find an integer n, $0 \leq n < N$ such that $h = g^n$.*

Along the years, and motivated by precise algorithms or cryptographic implementations, variants of this basic problem appeared. For instance:

Interval-DLP (IDLP): *given $h \in G$, find an integer n, $a \leq n < b$ such that $h = g^n$.*

We may also need to compute multiple instances of the DLP:

Batch-DLP (BDLP): *given $\{h_1, \ldots, h_k\} \subset G$, find integers n_i's, $0 \leq n_i < N$ such that $h_i = g^{n_i}$.*

A variant of this is **Delayed target DLP** where we can precompute many logs before receiving the actual target. This was used in the *logjam attack* where logarithms could be computed in real time during an SSL connection [9].

We may relate them to Diffie-Hellman problems, such as

Computational DH problem (DHPor CDH): given (g, g^a, g^b), compute g^{ab}.

Decisional DH problem (DDHP): given (g, g^a, g^b, g^c), do we have $c = ab \bmod N$?

We use $A \leq B$ to indicate that A is easier than B (i.e., there is polynomial time reduction from B to A). The first easy result is the following:

Proposition 9.1. DDHP $\leq$ DHP $\leq$ DLP.

In some cases, partial or heuristic reciprocals have been given, most notably in [86, 87]. With more and more applications and implementations available of different DH bases protocols, more problems arose, notably related to static variants. We refer to [77] for a survey.

9.1.2 The Pohlig-Hellman Reduction

In this section, we reduce the cost of DLP in a group of size N to several DLPswhose overall cost is dominated by that of the DLP for the largest $p \mid N$.

Proposition 9.2. *Let G be a finite cyclic group of order N whose factorization into primes is known,*

$$N = \prod_{i=1}^{r} p_i^{\alpha_i}$$

where the p_i's are all distinct. Then DLP in G can be solved using the DLPson all subgroups of order $p_i^{\alpha_i}$.

Proof. Solving $g^x = a$ is equivalent to finding $x \bmod N$, i.e., $x \bmod p_i^{\alpha_i}$ for all i, using the Chinese remaindering theorem.

Suppose $p^\alpha \mid\mid N$ (which means $p^\alpha \mid N$ but $p^{\alpha+1} \nmid N$) and $m = N/p^\alpha$. Then $b = a^m$ is in the cyclic group of order p^α generated by $h = g^m$. We can find the log of b in this group, which yields $x \bmod p^\alpha$. From this, we have reduced DLP to r DLPs in smaller cyclic groups.

How do we proceed? First compute $c_1 = b^{p^{\alpha-1}}$ and $h_1 = h^{p^{\alpha-1}}$. Both elements belong to the cyclic subgroup of order p of G, generated by h_1. Writing $y = x \bmod p^\alpha = y_0 + y_1 p + \cdots + y_{\alpha-1}p^{\alpha-1}$ with $0 \le y_i < p$, we see that $y = \log_h(b) \bmod p^\alpha$. We compute

$$h_1^y = h_1^{y_0}$$

so that y_0 is the discrete logarithm of c in base h_1. Write

$$c_2 = b^{p^{\alpha-2}} = h_1^{(y_0+y_1 p)p^{\alpha-2}}$$

or

$$c_2 h_1^{-y_0 p^{\alpha-2}} = h_1^{y_1}.$$

In this way, we recover y_1 by computing the discrete logarithm of the left hand side w.r.t. h_1 again. □

We have shown that DLP in a cyclic group of order p^α can be replaced by α solutions of DLP in a cyclic group of order p and some group operations. If p is small, all these steps will be easily solved by table lookup. Otherwise, the methods presented in the next section will apply and give a good complexity.

A direct cryptographic consequence is that for cryptographic use, N must have at least one large prime factor.

9.1.3 A Tour of Possible Groups

Easy Groups

Let us say a group is *easy* for DL if DLP can be solved in polynomial time for this group. DLP is easy in $(\mathbb{Z}/N\mathbb{Z}, +)$, since $h = ng \bmod N$ is solvable in polynomial time (Euclid). This list was recently enlarged to cases where the discrete log can be computed in quasi-polynomial time [17].

As for algebraic curves, supersingular elliptic curves were shown to be somewhat weaker in [89]; the same is true for hyperelliptic curves [49, 37]. Elliptic curves of trace 1 (also called *anomalous curves*) were shown to be easy in [109, 104]; the result was extended to hyperelliptic anomalous curves in [99].

Not-so-easy groups

Relatively easy groups are those for which subexponential methods exist: finite fields (of medium or large characteristic), algebraic curves of very large genus, class groups of number fields.

Probably difficult groups, for which we know of nothing but exponential methods, include elliptic curves (see [50] for a recent survey) and curves of genus 2.

9.2 Generic Algorithms

We start with algorithms solving DLP in a generic way, which amounts to saying that we use group operations only. We concentrate on generic groups. For particular cases, as elliptic curves, we refer to [53] for optimized algorithms. We emphasize that generic methods are the only known methods for solving the DLP over ordinary elliptic curves.

The chosen setting is that of a group $G = \langle g \rangle$ of prime order N, following the use of the Pohlig-Hellman reduction. Enumerating all possible powers of g is an $O(N)$ process, and this is enough when N is small. Other methods include Shanks's and Pollard's, each achieving a $O(\sqrt{N})$ time complexity, but different properties as determinism or space. We summarize this in Table 9.1. The baby-steps giant-steps (BSGS) method and its variants are deterministic, whereas the other methods are probabilistic. It is interesting to note that Nechaev and Shoup have proven that a lower bound on generic DLP (algorithms that use group operations only) is precisely $O(\sqrt{N})$ (see for instance [111]).

Due to the large time complexity, it is desirable to design distributed versions with a gain of p in time when p processors are used. This will be described method by method.

Table 9.1: Properties of generic DL algorithms.

Algorithm	group	interval	time	space
BSGS	✓	✓	$O(\sqrt{N})$	$O(\sqrt{N})$
RHO	✓	–	$O(\sqrt{N})$	$O(\log N)$
Kangaroo	✓	✓	$O(\sqrt{N})$	$O(\log N)$

9.2.1 Shanks's Baby-Steps Giant-Steps Algorithm

Shanks's idea, presented in the context of class groups [107], became a fundamental tool for operating in generic groups, for order or discrete logarithm computations. All variants of it have a time complexity $O(\sqrt{N})$ group operations for an $O(\sqrt{N})$ storage of group elements. They all differ by the corresponding constants and scenarii in which they are used. From [53], we extract Table 9.2.

Table 9.2: Table of constants C such that the complexity is $C\sqrt{N}$.

Algorithm	Average-case time	Worst-case time
BSGS	1.5	2.0
BSGS optimized for av. case	1.414	2.121
IBSGS	1.333	2.0
Grumpy giants	1.25?	≤ 3
RHO with dist. pts	$1.253(1 + o(1))$	∞

The goal of this subsection is to present the original algorithm together with some of its more recent variants. We refer to the literature for more material and analyses.

Original algorithm and analysis

The standard algorithm runs as follows. Write the unknown n in base u for some integer u that will be precised later on:

$$n = cu + d, 0 \leq d < u, \quad 0 \leq c < N/u.$$

We rewrite our equation as

$$g^n = h \Leftrightarrow h \circ (g^{-u})^c = g^d.$$

The algorithm is given in Figure 9.1. It consists of evaluating the right-hand side for all possible d by increment of 1 (baby steps), and then computing all left-hand sides by increment of u (giant steps), until a match is found.

Algorithm 9.1: Baby-steps giant-steps.

Function *BSGS(G, g, N, h)*
Input : $G \supset \langle g \rangle$, g of order N; $h \in \langle g \rangle$
Output : $0 \leq n < N$, $g^n = h$
$u \leftarrow \left\lceil \sqrt{N} \right\rceil$;
`// Step 1 (baby steps)`
initialize a table $\mathcal{B}$ for storing u pairs (element of G, integer $< N$);
store($\mathcal{B}$, $(1_G, 0)$);
$H \leftarrow g$; store($\mathcal{B}$, $(H, 1)$);
for $d := 2$ **to** $u - 1$ **do**
  $H \leftarrow H \circ g$;
  store($\mathcal{B}$, (H, d));
end
`// Step 2 (giant steps)`
$H \leftarrow H \circ g$;
$f \leftarrow 1/H = g^{-u}$;
$H \leftarrow h$;
for $c := 0$ **to** N/u **do**
  `//` $H = h \circ f^c$
  if $\exists (H', d) \in \mathcal{B}$ *such that* $H = H'$ **then**
    `//` $H = h \circ f^c = g^d$ `hence` $n = cu + d$
    return $cu + d$;
  end
  $H \leftarrow H \circ f$;
end

The number of group operations is easily seen to be $C_o = u + N/u$ in the worst case, minimized for $u = \sqrt{N}$, leading to a deterministic complexity of $2\sqrt{N}$ group operations. On average, c will be of order $N/(2u)$, so that the average cost is $(1 + 1/2)u$, which gives us the first entry in Table 9.2.

Step 1 requires u insertions in the set $\mathcal{B}$ and Step 2 requires N/u membership tests in the worst case. This explains why we should find a convenient data structure for $\mathcal{B}$ that has the smallest time for both operations. This calls for $\mathcal{B}$ to be represented by a hash table of some sort. The cost of these set operations will be $(u + N/u)O(1)$, again minimized by $u = \sqrt{N}$.

Remarks.

Variants exist when inversion has a small cost compared to multiplication, leading to writing $n = cu + d$ where $-c/2 \leq d < c/2$, thereby gaining in the constant in front of $\sqrt{N}$ (see [53] for a synthetic view). Cases where the distribution of parameters is non-uniform are studied in [26].

All kinds of trade-offs are possible if low memory is available. Moreover, different variants of BSGS exist when one wants to run through the possible steps in various orders (see [110]). If no bound is known on N, there are slower incremental algorithms that will find the answer; see [113] and [110].

As a final comment, BSGS is easy to distribute among processors with a shared memory.

Solving IDLP

BSGS works if we have a bound on N only. It also works when x is known to belong to some interval $[a, b[\subset [0, N[$. The process amounts to a translation of x to $[0, b - a[$ of length $b - a$ and therefore BSGS will require time $O(\sqrt{b - a})$.

Optimizing BSGS on average

On average, we could anticipate c to be around $N/2$, so that we may want to optimize the mean number of operations of BSGS, or $C_m = u + (N/2)/u$, leading to $u = \sqrt{N/2}$ and $C_m = \sqrt{2N}$, decreasing the memory used by the same quantity. The number of set operations also decreases. This gives us the line for BSGS optimized for average-case. Algorithm 9.1 is usable *mutatis mutandis*.

Interleaving baby steps and giant steps

Pollard [98] proposed a variant of the BSGS algorithm interleaving baby steps and giant steps, in order to decrease the average cost of BSGS. The idea is the following: If $x = cu + d$, we may find c and d after $\max(c, d)$ steps using the following algorithm. The rationale for the choice of u will be explained next.

Algorithm 9.2: Interleaved Baby steps-giant steps.

Function *IBSGS(G, g, N, h)*
Input : $G \supset \langle g \rangle$, g of order N; $h \in \langle g \rangle$
Output : $0 \leq n < N$, $g^n = h$
$u \leftarrow \left\lceil \sqrt{N} \right\rceil$;
initialize two tables $\mathcal{B}$ and $\mathcal{G}$ for storing u pairs (element of G, integer $< N$);
$H \leftarrow 1_G$; store($\mathcal{B}$, $(1_G, 0)$); store($\mathcal{G}$, $(1_G, 0)$);
$F \leftarrow h$;
$f \leftarrow g^{-u} = 1/g^u$;
for $i := 1$ **to** u **do**
  $H \leftarrow H \circ g$;
  if $\exists (H', c) \in \mathcal{G}$ *such that* $H = H'$ **then**
    // $H = g^i = H' = h \circ g^{-uc}$ `hence` $n = cu + i$
    return $cu + i$;
  end
  store($\mathcal{B}$, (H, i));
  $F \leftarrow F \circ f$;
  if $\exists (H', d) \in \mathcal{B}$ *such that* $F = H'$ **then**
    // $F = h \circ g^{-ui} = H' = g^d$ `hence` $n = iu + d$
    return $iu + d$;
  end
  store($\mathcal{G}$, (F, i));
end

First of all, remark that any integer n in $[0, N[$ may be written as $cu + d$ where $0 \leq c, d < u$. Algorithm 9.2 performs $2\max(c, d)$ group operations. We need to evaluate the average value of this quantity over the domain $[0, u[\times[0, u[$, which is equivalent to computing

$$\int_{x=0}^{1} \int_{y=0}^{1} \max(x, y) \, \mathrm{d}\, x \, \mathrm{d}\, y.$$

Fixing x, we see that $\max(x, y) = x$ for $y \leq x$ and y otherwise. Therefore the double integral is

$$\int_{x=0}^{1} \left(\int_{y=0}^{x} x \mathrm{d}\, y + \int_{y=x}^{1} y \mathrm{d}\, y \right) \mathrm{d}\, x = \frac{2}{3}.$$

We have proven that the mean time for this algorithm is $4/3\sqrt{N}$, hence a constant that is smaller than $\sqrt{2}$ for the original algorithm.

Grumpy giants

In [23], the authors designed a new variant of BSGS to decrease the average case running time again. They gave a heuristic analysis of it. This was precised and generalized to other variants (such as using negation) in [53]. We follow the presentation therein. For $u = \lceil \sqrt{N}/2 \rceil$, the algorithm computes the three sets of cardinality L that will be found later:

$$\mathcal{B} = \{g^i \text{ for } 0 \leq i < L\},$$
$$\mathcal{G}_1 = \{h \circ (g^{ju}) \text{ for } 0 \leq j < L\},$$
$$\mathcal{G}_2 = \{: h^2 \circ g^{-k(u+1)} \text{ for } 0 \leq k < L\},$$

and waits for a collision between any of the two sets, in an interleaved manner. The algorithm succeeds when one of the following sets contains the discrete logarithm we are looking for:

$$\begin{aligned} \mathcal{L}_L &= \{i - ju \pmod N, 0 \leq i, j < L\} \\ &\cup \{2^{-1}(i + k(u+1)) \pmod N, 0 \leq i, k < L\} \\ &\cup \{ju + k(u+1) \pmod N, 0 \leq j, k < L\}. \end{aligned}$$

For ease of exposition of Algorithm 9.3, we define $\mathrm{Expo}(u, j)$ to be the exponent of g in case of $\mathcal{G}_j$ for $j = 1..2$. Precisely, a member of $\mathcal{G}_j$ is

$$h^j \circ f_j = h^j \circ g^{\mathrm{Expo}(u,j)},$$

with $\mathrm{Expo}(u, j) = (-1)^{j-1}(u + j - 1)$.

It is conjectured that u is optimal and that L can be taken as $O(\sqrt{N})$. Experiments were carried out to support this claim in [23, 53]. Moreover [53] contains an analysis of the algorithm, leading to $1.25\sqrt{N}$ as total group operations.

9.2.2 The RHO Method

The idea of Pollard was to design an algorithm solving DLP for which the memory requirement would be smaller than that of BSGS. Extending his RHO method of factoring, he came up with the idea of RHO for computing discrete logarithms [97].

A basic model

Let E be a finite set of cardinality m and suppose we draw uniformly n elements from E with replacement. The probability that all n elements are distinct is

Theorem 9.3.

$$\mathrm{Proba} = \frac{1}{m} \prod_{k=1}^{n-1} \left(1 - \frac{k}{m}\right).$$

Taking logarithms, and assuming $n \ll m$, we get

$$\log \mathrm{Proba} \approx \log(n/m) - \frac{n(n-1)}{2m}.$$

This means that taking $n = O(\sqrt{m})$ will give a somewhat large value for this probability.

We can derive from this a very simple algorithm for computing discrete logarithms, presented as Algorithm 9.4. Its time complexity would be $O(\sqrt{m} \log m)$ on average, together with a space $O(\sqrt{m})$, which is no better than BSGS.

If we assume that N is prime, the only case where $v - v'$ is non-invertible is that of $v = v'$. In that case, we hit a useless relation between g and h that is discarded.

Our basic model is highly distributable. Unfortunately, the memory problem is still there. It does not solve the space problem, so that we have to replace this by deterministic random walks, as explained now.

Algorithm 9.3: Two grumpy giants and a baby.

Function *Grumpy(G, g, N, h)*
Input : $G \supset \langle g \rangle$, g of order N; $h \in \langle g \rangle$
Output : $0 \leq n < N$, $g^n = h$
$u \leftarrow \left\lceil \sqrt{N} \right\rceil$;
initialize three tables $\mathcal{B}$, $\mathcal{G}_1$ and $\mathcal{G}_2$ for storing u pairs (element of G, integer $< N$);
$H \leftarrow 1_G$; store($\mathcal{B}$, $(H, 0)$);
$F_1 \leftarrow h$; store($\mathcal{G}_1$, $(F_1, 0)$);
$F_2 \leftarrow h^2$; store($\mathcal{G}_2$, $(F_2, 0)$);
$f_1 \leftarrow g^u$;
$f_2 \leftarrow 1/(f_1 \circ g) = 1/g^{u+1}$;
for $i := 1$ **to** L **do**
  $H \leftarrow H \circ g$;
  for $j := 1$ **to** 2 **do**
    if $\exists (H', c) \in \mathcal{G}_j$ *such that* $H = H'$ **then**
      // $H = g^i = H' = h^j \circ f_j^c$
      return $(-\text{Expo}(u, j)c + i)/j \pmod N$;
    end
  end
  store($\mathcal{B}$, (H, i));
  for $j := 1$ **to** 2 **do**
    $F_j \leftarrow F_j \circ f_j$;
    if $\exists (H', d) \in \mathcal{B}$ *such that* $F_j = H'$ **then**
      // $F_j = h^j \circ g^{ui} = H' = g^d$
      return $(-\text{Expo}(u, j)i + d)/j \pmod N$;
    end
    $j' \leftarrow 3 - j$;
    if $\exists (H', c) \in \mathcal{G}_{j'}$ *such that* $F_j = H'$ **then**
      // $F_j = h^j \circ f_j^i = H' = h^{j'} \circ f_{j'}^c$
      return $(\text{Expo}(u, j')c - \text{Expo}(u, j)i + d)/(j - j') \pmod N$;
    end
    store($\mathcal{G}_j$, (F_j, i));
  end
end

Algorithm 9.4: Naive DLP algorithm.

Function *NaiveDL(G, g, N, h)*
  Input : $G \supset \langle g \rangle$, g of order N; $h \in \langle g \rangle$
  Output : $0 \leq n < N$, $g^n = h$
  initialize a table $\mathcal{L}$ for storing u triplets (element of G, two integers $< N$);
  repeat
    draw u and v at random modulo N;
    $H \leftarrow g^u \circ h^v$;
    if $\exists (H', u', v') \in \mathcal{L}$ *such that* $H = H'$ **then**
      // $H = g^u \circ h^v = g^{u'} \circ h^{v'}$ hence $n(v - v') = u' - u$
      if $v - v'$ *is invertible modulo* N **then**
        return $(u' - u)/(v - v') \bmod N$;
      end
    end
    else
      store($\mathcal{L}$, (H, u, v));
    end
  until *a collision is found*;

Functional digraphs

Consider E of cardinality m as above and let $f : E \to E$ be a function on E. Consider the sequence $X_{n+1} = f(X_n)$ for some starting point $X_0 \in E$. The *functional digraph* of X is built with vertices X_i's; an edge is put between X_i and X_j if $f(X_i) = X_j$. Since E is finite, the graph has two parts, as indicated in Figure 9.1.

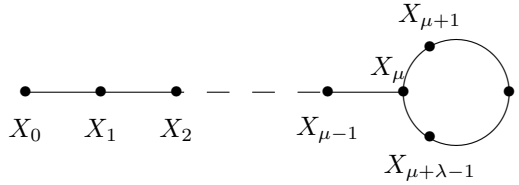


Figure 9.1: Functional digraph.

Since E is finite, the sequence X must end up looping. The first part of the sequence is the set of X_i's that are reached only once and there are μ of them; the second part forms a loop containing λ distinct elements.

Examples. 1) $E = G$ is a finite group, we use $f(x) = ax$, and $x_0 = a$, (x_n) purely is periodic, i.e., $\mu = 0$, and $\lambda = \mathrm{ord}_G(a)$.
2) Take $E_m = \mathbb{Z}/11\mathbb{Z}$ and $f : x \mapsto x^2 + 1 \bmod 11$: We give the complete graph for all possible starting points in Figure 9.2. The shape of it is quite typical: a cycle and trees plugged on the structure.

By Theorem 9.3, λ and μ cannot be too large on average, since $n = \lambda + \mu$. A convenient source for all asymptotic complexities of various parameters of the graph can be found in [46]. In particular:

Theorem 9.4. *When* $m \to \infty$

$$\overline{\lambda} \sim \overline{\mu} \sim \sqrt{\frac{\pi m}{8}} \approx 0.627\sqrt{m}.$$

Finding λ and μ is more easily done using the notion of *epact*.

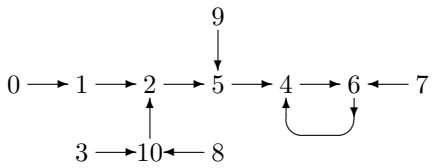


Figure 9.2: The functional digraph of $f : x \mapsto x^2 + 1 \bmod 11$.

Proposition 9.5. *There exists a unique $e > 0$ (epact) s.t. $\mu \leq e < \lambda + \mu$ and $X_{2e} = X_e$. It is the smallest non-zero multiple of λ that is $\geq \mu$: If $\mu = 0$, $e = \lambda$ and if $\mu > 0$, $e = \lceil \frac{\mu}{\lambda} \rceil \lambda$.*

Proof. The equation $X_i = X_j$ with $i < j$ only if i and j are larger than or equal to μ. Moreover, λ must divide $j - i$. If we put $i = e$ and $j = 2e$, then $\mu \leq e$ and $\lambda \mid e$. There exists a single multiple of λ in any interval of length λ, which gives unicity for e. When $\mu = 0$, it is clear that the smallest $e > 0$ must be λ. When $\mu > 0$, the given candidate satisfies all the properties. □

From [46], we extract

Theorem 9.6. $\overline{e} \sim \sqrt{\frac{\pi^5 m}{288}} \approx 1.03\sqrt{m}$.

which means that finding the epact costs $O(\sqrt{m})$ with a constant not too large compared to the actual values of μ and λ. Note that in most cryptographic applications, the collision $x_{2e} = x_e$ will be enough to solve our problem.

From a practical point of view, a nice and short algorithm by Floyd can be used to recover the epact and is given as Algorithm 9.5. We need $3e$ evaluations of f and e comparisons. Ideas for decreasing the number of evaluations are given in [31] (see also [91] when applied to integer factorization).

Algorithm 9.5: Floyd's algorithm.

```
Function epact(f, x_0)
    Input  : A function f, a starting point x_0
    Output : The epact of (x_n) defined by x_{n+1} = f(x_n)
    x ← x_0; y ← x_0; e ← 0;
    repeat
        e ← e + 1;
        x ← f(x);
        y ← f(f(y));
    until x = y;
    return e.
```

More parameters can be studied and their asymptotic values computed. Again, we refer to [46], from which we extract the following complements.

Theorem 9.7. *The expected values of some of the parameters related to the functional graph G are*

- *the number of components is $\frac{1}{2} \log m$;*
- *the component size containing a node $\nu \in G$ is $2m/3$;*
- *the tree size containing ν is $m/3$ (maximal tree rooted on a circle containing ν);*

- *the number of cyclic nodes is* $\sqrt{\pi m/2}$ *(a node is* cyclic *if it belongs to a cycle).*

A way to understand these results is to imagine that there is a giant component that contains almost all nodes.

Discrete logarithms

The idea of Pollard is to build a function f from G to G appearing to be random, in the sense that the epact of f is $c\sqrt{N}$ for some small constant c. This can be realized via multiplications by random points and/or perhaps squarings in G.

Building on [105], Teske [114] has suggested the following: precompute r random elements $z_i = g^{\gamma_i} \circ h^{\delta_i}$ for $1 \leq i \leq r$ for some random exponents. Then use some hash function $\mathcal{H} : G \to \{1, \ldots, r\}$. Finally, define $f(y) = y \circ z_{\mathcal{H}(y)}$. The advantage of this choice is that we can represent any iterate x_i of f as

$$x_i = g^{c_i} \circ h^{d_i},$$

where (c_i) and (d_i) are two integer sequences. When e is found:

$$g^{c_{2e}} \circ h^{d_{2e}} = g^{c_e} \circ h^{d_e}$$

or

$$g^{c_{2e}-c_e} = h^{d_e-d_{2e}},$$

i.e.,

$$n(c_{2e} - c_e) \equiv (d_e - d_{2e}) \bmod N.$$

With high probability, $c_{2e} - c_e$ is invertible modulo N and we get the logarithm of h. When we hit a collision and it is trivial, it is no use continuing the algorithm.

Experimentally, $r = 20$ is enough to have a large mixing of points. Under a plausible model, this leads to a $O(\sqrt{N})$ method (see [114]). We give the algorithm in Algorithm 9.6. As an example, if G contains integers, we may simply use $\mathcal{H}(x) = 1 + (x \bmod r)$.

Parallel RHO

How would one program a parallel version of RHO? We have to modify the algorithm. First of all, we cannot use the notion of epact any more. If we start p processors on finding the epact of their own sequence, we would not gain anything, since all epacts are of the same (asymptotic) size. We need to share computations.

The idea is to launch p processors on the same graph with the same iteration function and wait for a collision. Since we cannot store all points, we content ourselves with *distinguished elements*, i.e., elements having a special form, uniformly with some probability θ over G. (For integers, one can simply decide that a distinguished integer is 0 modulo a prescribed power of 2.) Each processor starts its own path from a random value in $\langle g \rangle$ and each time it encounters a distinguished element, it compares it with shared distinguished elements already found and when a useful collision is found, the program stops. The idea is that two paths colliding at some point will eventually lead to the same distinguished element, that will be found a little while later (see Figure 9.3). Typically, if $\theta < 1$ is the proportion of distinguished elements, the time to reach one of these will be $1/\theta$. Remembering properties of functional digraphs, this probability should satisfy $1/\theta < c\sqrt{N}$ for some constant $c > 0$.

In view of Theorem 9.7, the method succeeds since there is a giant component in which the processors have a large probability to run in. At worst, we would need $O(\log m)$ of these to be sure to have at least two processors in the same component.

There are many fine points that must be dealt with in an actual implementation. For ease of reading, we first introduce a function that computes a *distinguished path*, starting

Algorithm 9.6: RHO algorithm.

Function *RHO(G, g, N, h, $\mathcal{H}$, $(z_i, \gamma_i, \delta_i)$)*
Input : $\mathcal{H} : G \to \{1, \ldots, r\}$; $(z_i)_{1 \le i \le r}$ random powers $z_i = g^{\gamma_i} \circ h^{\delta_i}$ of G
Output : $0 \le n < N$, $g^n = h$
if $h = 1_G$ **then**
  return 0
end
// `invariant:` $x = g^{u_x} \circ h^{v_x}$
$x \leftarrow h$; $u_x \leftarrow 0$; $v_x \leftarrow 1$;
$y \leftarrow x$; $u_y \leftarrow u_x$; $v_y \leftarrow v_x$;
repeat
  $(x, u_x, v_x) \leftarrow$ Iterate(G, N, $\mathcal{H}$, $(z_i, \gamma_i, \delta_i)$, x, u_x, v_x);
  $(y, u_y, v_y) \leftarrow$ Iterate(G, N, $\mathcal{H}$, $(z_i, \gamma_i, \delta_i)$, y, u_y, v_y);
  $(y, u_y, v_y) \leftarrow$ Iterate(G, N, $\mathcal{H}$, $(z_i, \gamma_i, \delta_i)$, y, u_y, v_y);
until $x = y$;
// $g^{u_x} \circ h^{v_x} = g^{u_y} \circ h^{v_y}$
if $v_x - v_y$ *is invertible modulo* N **then**
  return $(u_y - u_x)/(v_x - v_y) \pmod N$;
end
else
  return *Failure*.
end

Algorithm 9.7: RHO iteration algorithm.

Function *Iterate(G, N, $\mathcal{H}$, $(z_i, \gamma_i, \delta_i)$, x, u_x, v_x)*
Input : $\mathcal{H} : G \to \{1, \ldots, r\}$; $(z_i)_{1 \le i \le r}$ random powers $z_i = g^{\gamma_i} \circ h^{\delta_i}$ of G;
$x = g^{u_x} \circ h^{v_x}$
Output : $f(x, u_x, v_x) = (w, u_w, v_w)$ such that $w = g^{u_w} \circ h^{v_w}$
$i \leftarrow \mathcal{H}(x)$;
return $(x \circ z_i, u_x + \gamma_i \pmod N, v_x + \delta_i \pmod N)$.

from a point and iterating until a distinguished element is reached, at which point is it returned.

At this point, the master can decide to continue from this distinguished element, or start a new path. One of the main problems we can encounter is that a processor be trapped in a (small) cycle. By the properties of random digraph, a typical path should be of length $O(\sqrt{N})$; if θ is small enough, the probability to enter a cycle will be small. However, in some applications, small cycles exist. Therefore, we need some cycle detection algorithm, best implemented using a bound on the number of elements found. Modifying Algorithm 9.8 can be done easily, for instance, giving up on paths with length $> 20/\theta$ as suggested in [118]. The expected running time is $\sqrt{\pi N/2}/p + 1/\theta$ group operations.

Note that in many circumstances, we can use an automorphism in G, and we take this into account for speeding up the parallel RHO process, despite some technical problems that arise (short cycles). See [44], and more recently [72, 24].

Other improvements are discussed in [36] for prime fields, with the aim of reducing the cost of the evaluation of the iteration function.

9.2.3 The Kangaroo Method

This method was designed to solve Interval-DLP with a space complexity as small as that of RHO, assuming the discrete logarithm we are looking for belongs to $[0, \ell]$ with $\ell \le N$. We

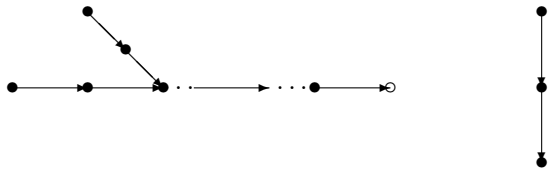


Figure 9.3: Paths.

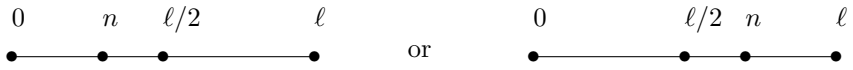
Algorithm 9.8: Finding a distinguished path.

Function *DistinguishedPath(f, x_0)*
Input : A function f, a starting point x_0
Output : The first distinguished element found starting at x_0,
$x \leftarrow x_0$; **repeat**
  $x \leftarrow f(x)$;
until x *is distinguished*;
return x.

would like to obtain an algorithm whose running time is $O(\sqrt{\ell})$ instead of $O(\sqrt{N})$.

The idea is to have two processes, traditionally called *tame kangaroo* and *wild kangaroo*. The tame kangaroo follows a random path starting from $g^{\ell/2}$ and adding random integers to the exponent, while the wild kangaroo starts from $h = g^n$ and uses the same deterministic random function. We use a sequence of integer increments $(\delta_i)_{1 \le i \le r}$ whose mean size is m. Then, we iterate: $f(x) = x \circ g^{\delta_{\mathcal{H}(x)}}$. Both kangaroos can be written $T = g^{d_T}$ and $W = h \circ g^{d_W}$ for two integer sequences d_T and d_W that are updated when computing f.

When hitting a distinguished element, it is stored in a list depending on its character (tame or wild). When a collision occurs, the discrete logarithm is found. The analysis is heuristic along the following way. The original positions of K_T and K_W can be either



In either case, we have a back kangaroo (B) and a front kangaroo (F) heading right. They are at mean mutual distance $\ell/4$ at the beginning. Since the average distance between two points is m, B needs $\ell/(4m)$ jumps to reach the initial position of F. After that, B needs m jumps to reach a point already reached by F. The total number of jumps is therefore $2(\ell/(4m) + m)$, which is minimized for $m = \sqrt{\ell}/2$, leading to a $2\sqrt{\ell}$ cost. A more precise analysis is given in [118]. The reader can find details as Algorithm 9.9.

A close algorithm that uses another model of analysis (though still heuristic) is that of Gaudry-Schost [54], improving on work by Gaudry and Harley. This algorithm is generalized to any dimension (e.g., solving $g^x = g^{a_1 n_1 + a_2 n_2 + \cdots + a_d n_d}$ for given (a_i)'s) and improved in [51] (see also [52] for the use of equivalence classes).

Parallel kangaroos

The idea, as for parallel RHO, is to start p kangaroos that will discover and store distinguished elements. Following [98], we assume $p = 4p'$, and select $u = 2p' + 1$, $v = 2p' - 1$, so that $p = u + v$. Increments of the jumps will be $(uvs_1, \ldots, uvs_k)$ for small s_i's, insisting on the mean to be $\approx \sqrt{\ell/(uv)}$. The i-th tame kangaroo will start at $g^{\ell/2+iv}$ for $0 \le i < u$; wild kangaroo W_i will start from $h \circ g^{iu}$, $0 \le i < v$. A collision will be $\ell/2 + iv = n + ju \bmod (uv)$ and the solution is unique. This prevents kangaroos of the same herd from colliding. The final running time is effectively divided by p.

Algorithm 9.9: Sequential kangaroos.

Function *Kangaroo(G, g, N, h, ℓ)*
Input : $G \supset \langle g \rangle$, g of order N; $h \in \langle g \rangle$
Output : $0 \leq n < \ell$, $g^n = h$
$m \leftarrow \left\lceil \sqrt{\ell}/2 \right\rceil$;
compute positive increments $(\delta_i)_{1 \leq i \leq r}$ of mean m;
initialize two tables $\mathcal{T}$ and $\mathcal{W}$ for storing pairs (element of G, integer $< N$);
$T \leftarrow g^{\ell/2}$; $d_T \leftarrow \ell/2$;
$W \leftarrow h$; $d_W \leftarrow 0$;
while *true* **do**
  $(T, d_T) \leftarrow f((\delta_i), T, d_T)$;
  if $\exists (W', d') \in \mathcal{W}$ *such that* $W' = T$ **then**
    // $T = g^{d_T}$, $W' = h \circ g^{d'}$
    return $(d_T - d') \pmod N$;
  end
  $(W, d_W) \leftarrow f((\delta_i), W, d_W)$;
  if $\exists (T', d') \in \mathcal{T}$ *such that* $T' = W$ **then**
    // $T' = g^{d'}$, $W = h \circ g^{d_W}$
    return $(d' - d_W) \pmod N$;
  end
end

9.2.4 Solving Batch-DLP

Using BSGS

If we know in advance that we have to solve the DLP for k instances, then Step 1 of BSGS is unchanged (but with another u) and Step 2 is performed at most kN/u times. This implies that we can minimize the total cost $u + kN/u$ using $u = \sqrt{kN}$, a gain of $\sqrt{k}$ compared to applying the algorithm k times. Mixing this with other tricks already mentioned is easy.

Parallel methods

The work of [45] was analyzed in [80]: A batch of k discrete logarithms in a group of order N reduces to an average $\Theta(k^{-1/2}N^{1/2})$ group operations for $k \ll N^{1/4}$; each DL costs $\Theta(N^{3/8})$. [80] also defines some problems related to DLP. The method was further studied in [60].

A more systematic way to consider the problem is the following; see [23] and its follow-up [22], where interval-batch-DLP is also considered and solved with the same ideas. We consider again a table of random steps not involving any target in its definition. The idea is to build a table T of distinguished elements found by random walks starting at random elements g^x. If T is the table size and W the length of the walk, then TW elements will be encountered. When given a target h, a random walk of length W will encounter one of the elements in T, solving DLP for h. The probability that none of the points in the new walk encounters any of the first is

$$\left(1 - \frac{1}{N}\right)^{TW^2}.$$

Taking logarithms, this is close to TW^2/N, so that a reasonable chance of success is for $W \approx \alpha\sqrt{N/T}$ for some constant α. Using this, the probability can be written $\exp(-\alpha^2)$, favoring rather large αs, therefore enabling (and favoring) parallel work too.

Extending this algorithm to the finding of k targets leads to a total cost of

$$O(TW) + kO(W) = O((T+k)W) = O(\sqrt{TN} + k\sqrt{N/T}),$$

and this is dominated by kW for $k > T$. If we want to optimize the cost as a function of k, we see that $T = k$ is minimal. For $T = N^{1/3}$, we get $W = N^{1/3}$ for each walk and $TW = N^{2/3}$ for the precomputation phase.

For a real implementation, we can choose $t = \lceil \log_2 N/3 \rceil$. If G contains integers, define x to be distinguished if $x \equiv 0 \bmod 2^t$. With this choice, we need to store 2^t elements; 2^{2t} operations are needed for the precomputation phase and 2^t for each of the 2^t target.

9.3 Finite Fields

9.3.1 Introduction

There exist dedicated algorithms to compute DL in finite fields that exploit the structure of the finite field. These algorithms are sub-exponential in the size of the field (but not in the involved subgroup order). Moreover, in 2014 two variants of a quasi-polynomial-time algorithm were proposed, for a class of finite fields such as $\mathbb{F}_{2^n}$ and $\mathbb{F}_{3^n}$, where n is composite. As a consequence, any pairing-friendly curve defined over a finite field of small characteristic should definitely be avoided. We explain in the following the main idea of these algorithms known as *index calculus methods*. Three main variants apply to three different types of finite fields (small, medium, and large characteristic, as explained below). We give the specializations of each variant and the range of finite fields to which they apply. Moreover when the prime defining the finite field is of special form (e.g., given by a polynomial), special variants provide an even lower complexity to compute DL.

Interest for pairing-based cryptography

Studying the complexity of DL computations in finite fields is essential for pairing-based cryptosystem designers. The pairing target group for any algebraic curve defined over a finite field is a subgroup in an extension of that finite field. Here are the most common finite fields that arise with pairing-friendly curves, in increasing order of hardness of DL computation:

1. $\mathbb{F}_{2^{4n}}$ and $\mathbb{F}_{3^{6m}}$ for supersingular curves defined over $\mathbb{F}_{2^n}$, $\mathbb{F}_{3^m}$, resp.
2. Small extension $\mathbb{F}_{p^n}$ of prime field with p of special form, given by a polynomial. This is the case for curves in families such as MNT [90], BLS [20], BN [21] curves, and any family obtained with the Brezing-Weng method [32].
3. Small extension of prime field $\mathbb{F}_p$, where p does not have any special form, e.g., $\mathbb{F}_{p^2}, \mathbb{F}_{p^6}$ for supersingular curves in large characteristic, and any curves generated with the Cocks-Pinch or Dupont-Enge-Morain methods.

The first class: Supersingular curves defined over a small characteristic finite field also correspond to a variant of the index calculus method where computing a DL is much easier. The two other classes are each parted in medium and large characteristic. This is explained in the next section.

Small, medium, and large characteristic

The finite fields are commonly divided into three cases, depending on the size of the prime p (the finite field characteristic) compared to the extension degree n, with $Q = p^n$. Each case has its own index calculus variant, and the most appropriate variant that applies qualifies the characteristic (as small, large, or medium):

- small characteristic: One uses the function field sieve algorithm, and the quasi-polynomial-time algorithm when the extension degree is suitable for that (i.e., smooth enough);
- Medium characteristic: one uses the NFS-HD algorithm. This is the High Degree variant of the Number Field Sieve (NFS) algorithm. The elements involved in the relation collection are of higher degree compared to the regular NFS algorithm.
- Large characteristic: one uses the Number Field Sieve algorithm.

Each variant (QPA, FFS, NFS-HD, and NFS) has a different asymptotic complexity. The asymptotic complexities are stated with the L-notation. This comes from the smoothness probability of integers. The explanation will be provided in Section 9.3.2. The L-notation is defined as follows.

Definition 9.8. Let Q be a positive integer. The L-notation is defined by

$$L_Q[\alpha, c] = \exp\Big(\big(c + o(1)\big)(\log Q)^{\alpha}(\log\log Q)^{1-\alpha}\Big) \quad \text{with } \alpha \in [0,1] \text{ and } c > 0 \ .$$

The α parameter measures the gap between polynomial time: $L_Q[\alpha = 0, c] = (\log Q)^c$, and exponential time: $L_Q[\alpha = 1, c] = Q^c$. When c is implicit, or obvious from the context, one simply writes $L_Q[\alpha]$. When the complexity relates to an algorithm for a prime field $\mathbb{F}_p$, one writes $L_p[\alpha, c]$.

Main historical steps

Here is a brief history of DL computations in finite fields. The DLP for cryptographic use was first stated in prime fields, without pairing context [43]. Binary fields were used for efficiency reasons (they provide a better arithmetic). Finite fields are not generic groups and there exist subexponential time algorithms to compute discrete logarithms in finite fields. About a century ago Kraitchik [78, pp. 119–123], [79, pp. 69–70, 216–267] introduced the *index calculus* method (from the French *calcul d'indice*) to compute discrete logarithms. His work was rediscovered in the cryptographic community in the 1970s. The first algorithms to compute discrete logarithms in prime fields are attributed to Adleman and Western–Miller [6, 119]. These algorithms had a complexity of $L_p[1/2, c]$. However Coppersmith showed as early as 1984 [40] that the DLP is much easier in binary fields than in prime fields of the same size. He obtained an $L_Q[1/3, c]$ running-time complexity for his algorithm. He also showed how fast his algorithm can compute discrete logarithms in $\mathbb{F}_{2^{127}}$ (his algorithm was better for fields whose extension degree is very close to a power of 2, in his record $127 = 2^7 - 1$). Later, Adleman and Huang generalized the Coppersmith's method to other small characteristic finite fields and named it the Function Field Sieve (FFS) algorithm [7, 8]. The asymptotic complexity was $L_Q[1/3, (\frac{32}{9})^{1/3} \approx 1.526]$. In 1986, the state-of-the-art for computing DL in prime fields was the Coppersmith, Odlyzko, and Schroeppel (COS) algorithm [42], in time $L_p[1/2, 1]$. Then in 1993, Gordon designed the Number Field Sieve algorithm for prime fields [55] and reached the same class of sub-exponential asymptotic complexity as the FFS algorithm: $L[1/3, c]$ but with a larger constant $c = 9^{1/3} \approx 2.080$. Between the two extremes ($\mathbb{F}_{2^n}$ and $\mathbb{F}_p$) is the medium characteristic case. In 2006, a huge work was done by Joux, Lercier, Smart, and Vercauteren [67] to propose algorithms to compute the DLP in $L_Q[1/3, c]$ for any finite field. Until 2013, the complexity formulas were frozen at this point:

- a complexity of $L_Q[1/3, (\frac{32}{9})^{1/3} \approx 1.526]$ for small characteristic finite fields with the Function Field Sieve algorithm [8];
- a complexity of $L_Q[1/3, (\frac{128}{9})^{1/3} \approx 2.423]$ for medium characteristic finite fields with the Number Field Sieve–High Degree algorithm [67];

- a complexity of $L_Q[1/3, (\frac{64}{9})^{1/3} \approx 1.923]$ for large characteristic finite fields with the Number Field Sieve algorithm [106, 67, 85].

It was not known until recently whether small characteristic fields could be a gap weaker than prime fields. However, computing power was regularly increasing and in 2012, Hayashi, Shimoyama, Shinohara, and Takagi were able to compute a discrete logarithm record in $\mathbb{F}_{3^{6\cdot 97}}$, corresponding to a 923-bit field [58], with the Function Field Sieve. Then in December 2012 and January 2013, Joux released two preprints later published in [63] with a $L_Q[1/4]$ algorithm, together with record-breaking discrete logarithms in $\mathbb{F}_{2^{4080}}$ and $\mathbb{F}_{2^{6168}}$. This was improved by various researchers. In 2014, Barbulescu, Gaudry, Joux, and Thomé [17] on one side and Granger, Kleinjung, and Zumbrägel [56] on the other side proposed two versions of a *quasi polynomial-time* algorithm (QPA) to solve the DLP in small characteristic finite fields. All the techniques that allowed this breakdown are not applicable to medium and large characteristic so far.

9.3.2 Index-Calculus Methods

We saw in Section 9.2 that the best complexity is obtained by balancing parameters in the algorithms. Over finite fields, FFS and NFS algorithms also reach sub-exponential complexity through balancing parameters.

Throughout this section, we follow the clear presentation of index-calculus methods made in [88]. We present in Algorithm 9.10 the basic index-calculus method for computing DL in a prime field.

Algorithm 9.10: Index calculus in $(\mathbb{Z}/p\mathbb{Z})^*$.

Function *IndexCalculus*($\mathbb{F}_p^*$, g, h)

Input : $\mathbb{F}_p^* \supset \langle g \rangle$, g of order dividing $p-1$

Output : $0 \leq x < p-1$, $g^x = h$

Phase 1: fix $\#\mathcal{B}$ and find the logarithms modulo $p-1$ of small primes in $\mathcal{B} = \{p_1, p_2, \ldots, p_{\#\mathcal{B}} \leq B\}$:

Choose integers $t_i \in [1, \ldots, p-1]$ s.t. g^{t_i} as an integer splits completely in small primes of $\mathcal{B}$:

$$g^{t_i} \bmod p = \prod_{b=1}^{\#\mathcal{B}} p_b^{\alpha_{b,i}}$$

so that taking the logarithm to the base g gives a *relation*:

$$t_i = \sum_{b=1}^{\#\mathcal{B}} \alpha_{b,i} \log_g p_b \bmod (p-1)$$

Phase 2: When enough relations are collected, solve the system to get $\{\log_g p_b\}_{1 \leq b \leq \#\mathcal{B}}$.

Phase 3: Compute the individual discrete logarithm of h in base g.

Look for t s.t. $hg^t \bmod p$ as an integer factors into small primes of $\mathcal{B}$:

$$hg^t \bmod p = \prod_{b=1}^{\#\mathcal{B}} p_b^{\alpha_b} \Leftrightarrow x + t \equiv \sum_{b=1}^{\#\mathcal{B}} \alpha_b \log_g p_b \bmod (p-1)$$

return x.

Example for a tiny prime p

Let $p = 1019$ and $g = 2$. We need to find the logarithms modulo 2 and 509, since $p - 1 = 2 \cdot 509$. We first locate some smooth values of $g^b \bmod p$:

$$2^{909} = 2 \cdot 3^2 \cdot 5, 2^{10} = 5, 2^{848} = 3^3 \cdot 5, 2^{960} = 2^2 \cdot 3.$$

The system to be solved is

$$\begin{pmatrix} 1 & 2 & 1 \\ 10 & 0 & 0 \\ 0 & 3 & 1 \\ 2 & 1 & 0 \end{pmatrix} \cdot X = \begin{pmatrix} 909 \\ 10 \\ 848 \\ 960 \end{pmatrix} \bmod 1018.$$

Solving modulo 2 and 509 separately and recombining by the Chinese remainder theorem, we find

$$\log_2 2 = 1, \log_2 3 = 958, \log_2 5 = 10.$$

Note that solving modulo 2 can be replaced by computations of Legendre symbols.

Consider computing $\log_2 314$. We find that

$$h \cdot g^{372} \equiv 2^4 \cdot 5^2 \bmod p$$

from which $\log_g h = 4 + 2 \cdot 10 - 372 \bmod 1018$ or $\log_2(314) = 670$. Had we used rational reconstruction (a classical trick for DL target solution), we would have found

$$h \cdot g^{409} \equiv 2/3 \bmod p$$

from which the log of t follows from

$$\log_g h + 409 \equiv 1 - \log_2 3 \bmod 1018.$$

A first complexity analysis

We will prove that:

Theorem 9.9. *The asymptotic heuristic running-time of Algorithm 9.10 is $L_p[1/2, 2]$ with $\#\mathcal{B} \approx B = L_p[1/2, 1/2]$.*

The running-time of algorithm 9.10 (p. 17) is related to smoothness probabilities of integers in Phase 1 and 3, and linear algebra in phase 2. We will use the L-notation formula given in 9.8. To estimate the smoothness probability, we need the result of Corollary 9.11 from Theorem 9.10.

Theorem 9.10 (Canfield–Erdős–Pomerance [34])**.** *Let $\psi(x, y)$ be the number of natural numbers smaller or equal to x which are y-smooth. If $x \geq 10$ and $y \geq \log x$, then it holds that*

$$\psi(x, y) = xu^{-u(1+o(1))} \text{ with } u = \frac{\log x}{\log y}\ , \tag{9.1}$$

where the limit implicit in the $o(1)$ is for $x \to \infty$.

The Canfield–Erdős–Pomerance [34] theorem provides a useful result to measure smoothness probability:

Corollary 9.11 (B-smoothness probability)**.** *For an integer S bounded by $L_Q[\alpha_S, \sigma]$ and a smoothness bound $B = L_Q[\alpha_B, \beta]$ with $\alpha_B < \alpha_S$, the probability that S is B-smooth is*

$$Pr[S \text{ is } B\text{-smooth}] = L_Q\left[\alpha_S - \alpha_B, -(\alpha_S - \alpha_B)\frac{\sigma}{\beta}\right]\ . \tag{9.2}$$

We will also need these formulas:

$$L_Q[\alpha, c_1]L_Q[\alpha, c_2] = L_Q[\alpha, c_1 + c_2] \text{ and } L_Q[\alpha, c_1]^{c_2} = L_Q[\alpha, c_1 c_2] \ . \tag{9.3}$$

Proof. (of Theorem 9.9.)

In our context, we want to find a smooth decomposition of the least integer r ($|r| < p$) s.t. $r = g^{b_i} \pmod p$. We need to estimate the probability of an integer smaller than p to be B-smooth. We write the formula, then balance parameters to ensure the optimal cost of this algorithm. Let us write the smoothness-bound B in sub-exponential form: $B = L_p[\alpha_B, \beta]$. Any prime in $\mathcal{B}$ is smaller than B. The probability of r bounded by $p = L_p[1, 1]$ to be B-smooth is

$$\Pr[r \text{ is } B\text{-smooth}] = L_p\left[1 - \alpha_B, -(1 - \alpha_B)\frac{1}{\beta}\right]^{1+o(1)} \ .$$

To complete Phase 2, we need enough relations to get a square matrix and solve the system, in other words, more than $\#\mathcal{B}$. Since the number of prime numbers $\leq B$ is $B/\log B$ for $B \to \infty$, we approximate $\#\mathcal{B} \approx B$. The number of iterations over g^{t_i} and smoothness tests to be made to get enough relations is

$$\text{number of tests} = \frac{\text{number of relations}}{B\text{-smoothness probability}} = \frac{B}{\Pr} = L_p[\alpha_B, \beta]L_p[1 - \alpha_B, (1 - \alpha_B)/\beta] \ .$$

The dominating α-parameter will be $\max(\alpha_B, 1 - \alpha_B)$ and is minimal for $\alpha_B = 1/2$. The number of tests is then $L_p[1/2, \beta + \frac{1}{2\beta}]$ (thanks to Equation (9.3)).

Now we compute the running time to gather the relations: This is the above quantity times the cost of a smoothing step. At the beginning of index calculus methods, a trial division was used, so one B-smooth test costs at most $\#\mathcal{B} = B = L_p[1/2, \beta]$ divisions. The total relation collection running time is then

$$L_p[1/2, 2\beta + 1/(2\beta)] \ .$$

The linear algebra phase finds the kernel of a matrix of dimension B. It has running-time of $B^\omega \approx L_p[1/2, \omega\beta]$ (ω is a constant, equal to 3 for classical Gauss, and nowadays we use iterative methods, of complexity $B^{2+o(1)}$; see Section 9.3.3). The total running time of the first two steps is

$$L_p[1/2, 2\beta + 1/(2\beta)] + L_p[1/2, 3\beta] \ .$$

The minimum of $\beta \mapsto 2\beta + \frac{1}{2\beta}$ is 2, for $\beta = 1/2$ (take the derivative of the function $x \mapsto 2x + \frac{1}{2x}$ to obtain its minimum, for $x > 0$). We conclude that the total cost of the first two phases is dominated by $L_p[1/2, 2]$: the relation collection phase.

The last phase uses the same process as finding one relation in Phase 1. It needs $1/\Pr[r = g^t \pmod p$ is B-smooth] tries of cost B each, hence $B/\Pr = L_p[1/2, \beta + \frac{1}{2\beta}] = L_p[1/2, 3/2]$. □

COS algorithm, Gaussian integer variant

Coppersmith, Odlyzko, and Schroeppel proposed in [42] a better algorithm by modifying the relation collection phase. They proposed to *sieve* over elements of well-chosen form, of size $\sim \sqrt{p}$ instead of p. They also proposed to use sparse matrix linear algebra, to improve the second phase of the algorithm. Computing the kernel of a sparse square matrix of size B has a running-time of $O(B^{2+o(1)})$ with their modified Lanczos algorithm. We present now their Gaussian Integer variant, so that the Gordon Number Field Sieve will be clearer in Section 9.3.4. We consider a generator g of $\mathbb{F}_p^*$ and want to compute the discrete logarithm x of h in base g, in the subgroup of $\mathbb{F}_p^*$ of prime order ℓ, with $\ell \mid p - 1$.

The idea of [42] is to change the relation collection: In the former case, iterating over g^{t_i} and taking the smallest integer $r \equiv g^{t_i} \pmod p$ always produces r of the same size as p.

In this version, another iteration is made. The idea is to produce elements r much smaller than p, to improve their smoothness probability. In the previous index calculus, we made relations between integers (we lifted $g^{t_i} \bmod p$ to $r \in \mathbb{Z}$). Here one side will consider integers, the second side will treat algebraic integers. Let A be a small negative integer which is a quadratic residue modulo p. Preferably, $A \in \{-1, -2, -3, -7, -11, -19, -43, -67, -163\}$ so that $\mathbb{Q}[\sqrt{A}]$ is a unique factorization domain. For ease of presentation, we assume that $p \equiv 1 \bmod 4$ and take $A = -1$. Our algebraic side will be the Gaussian integer ring $\mathbb{Z}[i]$. Now let $0 < U, V < \sqrt{p}$ such that $p = U^2 + V^2$ (computed via the rational reconstruction method, for example). The element U/V is a root of $x^2 + 1$ modulo p. For an analogy with the number field sieve, one can define $f = x^2 + 1$ for the first (algebraic) side and $g = U - xV$ for the second (rational) side. The two polynomials have a common root U/V modulo p. We define a map from $\mathbb{Z}[i]$ to $\mathbb{F}_p$:

$$\begin{array}{rrcl} \rho: & \mathbb{Z}[i] & \to & \mathbb{F}_p \\ & i & \mapsto & UV^{-1} \bmod p \\ \text{hence} & a - bi & \mapsto & V^{-1}(aV - bU) \bmod p\ . \end{array} \tag{9.4}$$

Now we sieve over pairs (a, b) on the rational side, looking for a B-smooth decomposition of the integer $aV - bU$, as in Algorithm 9.10. What will be the second member of a relation? Here comes the algebraic side. We consider the elements $a - bi \in \mathbb{Z}[i]$ (with the same pairs (a, b)) and iterate over them such that $a - bi$, as an ideal of $\mathbb{Z}[i]$, factors into prime ideals $\mathfrak{p}$ of $\mathbb{Z}[i]$ of norm $\mathcal{N}_{\mathbb{Z}[i]/\mathbb{Z}}(\mathfrak{p})$ smaller than B. (For example: $1 + 3i = (1 + i)(2 + i)$ with $\mathcal{N}(1 + 3i) = 1^2 + 3^2 = 10 = 2 \cdot 5 = \mathcal{N}(1 + i)\mathcal{N}(2 + i)$). Here is the magic: Since $\mathcal{N}_{\mathbb{Z}[i]/\mathbb{Z}}(a - bi) = a^2 + b^2$ and we sieve over small $0 < a < E$, $-E < b < E$, the norm of $a - bi$ will be bounded by E^2 and the product of the norms of the prime ideals $\mathfrak{p}$ in the factorization, which is equal to $a^2 + b^2$, will be bounded by E^2 as well. At this point, we end up with pairs (a, b) such that

$$aV - bU = \prod_{p_b \le B} p_b^{s_j}, \quad \text{and} \quad a - bi = \prod_{\mathcal{N}(\mathfrak{p}_{b'}) \le B} \mathfrak{p}_{b'}^{t_j}\ .$$

Then we use the map ρ to show up an equality, then get a relation. We have $\rho(a - bi) = a - bUV^{-1} = V^{-1}(aV - bU)$ so up to a factor V (which is constant along the pairs (a, b)), we have:

$$aV - bU = \prod_{p_b \le B} p_b^{s_j} = V \prod_{\mathcal{N}(\mathfrak{p}_{b'}) \le B} \rho(\mathfrak{p}_{b'})^{t_j}\ . \tag{9.5}$$

Here we don't need to know explicitly the value of $\rho(\mathfrak{p}_{b'})$ in $\mathbb{F}_p$. We simply consider it as an element of the basis $\mathcal{B}$ of small elements: $\mathcal{B} = \{V\} \cup \{p_b \le B\} \cup \{\rho(\mathfrak{p}_{b'}) :\ \mathcal{N}_{\mathbb{Z}[i]/\mathbb{Z}}(\mathfrak{p}_{b'}) \le B\}$. In the COS algorithm, the matrix is indeed two times larger than in the basic version of Algorithm 9.10 ($2B$ instead of B), but with norms $a^2 + b^2$ and $Va - bU$ much smaller; we can decrease the smoothness bound B. Taking the logarithm of Equation (9.5), we obtain an equation between logarithms of elements in $\mathcal{B}$:

$$\sum_{p_b \le B} s_j \log p_b = \log V + \sum_{\mathcal{N}_{\mathbb{Z}[i]/\mathbb{Z}}(\mathfrak{p}_{b'}) \le B} t_j \log \rho(\mathfrak{p}_{b'})\ . \tag{9.6}$$

The optimal choice of parameters is $E = B = L_p[1/2, 1/2]$, so that both sieving and linear algebra cost $L_p[1/2, 1]$, better than the previous $L_p[1/2, 2]$ thanks to the much better smoothness probabilities of the elements considered. Phase 2 of the algorithm computes the kernel of a large sparse matrix of more than $2B$ rows. Its expected running time is $(2B)^2$, hence again $L_p[1/2, 1]$. The expected running time of the individual logarithm computation (Phase 3) is $L_p[1/2, 1/2]$ ([42, §7]).

9.3.3 Linear Algebra

Before diving into the explanation of the most powerful algorithms for computing DLs, we make a pause and give some ideas on a technical but crucial problem: linear algebra and how we solve the systems arising in DL computations.

At the beginning of index calculus methods, the only available method was the ordinary Gauss algorithm, whose complexity is cubic in the number of rows (or columns) N of the matrix M. For small matrices, this is enough, but remember that entries in the matrix are elements of some finite field $\mathbb{F}_\ell$ for some large ℓ, in contrast with matrices we encounter in integer factorization, which are boolean. Fill-in in DL matrices is therefore extremely costly and we should be careful in doing this.

DL-matrices are very sparse, as factorization matrices. The coefficients are small integers (in absolute value). Special methods have been designed for all these matrices. Generally, some form of sparse Gaussian elimination is done in order to reduce the size of the matrix prior to the use of more sophisticated methods to be described below. The idea is to perform elimination using a sparse structure and minimize fill-in as long as possible. The general term for these is *filtering*, and it was optimized and made necessary due to the use of many large primes in recent years (see [30] for recent progress in the field).

Adaptation of numerical methods was done: The Lanczos iterative algorithm could be generalized to the finite field case. A new class of iterative methods was invented by Wiedemann [120]. Both classes have a complexity of $O(N^{2+\varepsilon})$ where we assume that our matrix has size $N^{1+\varepsilon}$. The core of the computation is the determination of the so-called Krylov subspaces, namely $Vect\langle M^i \cdot b\rangle$ for some fixed vector b. Applying M to b costs $O(N^{1+\varepsilon})$, and N iterations are needed. The advantage of such methods is also to be able to handle sparse structures for M.

Variants operating on blocks of vectors were designed by Coppersmith [39], for integer factorization as for DLP. Despite the use of such methods, space and time become quite a problem in recent records. The natural idea is to try to distribute the computations over clusters or larger networks. The only method that can be partly distributed is the block Wiedemann algorithm. A good reference for this part is Thomé's thesis [117]. All his work is incorporated and available in the CADO-NFS package [112], including the many refinements to distribute the computations over the world and some special tricks related to Schirokauer maps (see [70] for an independent work on the same topic). Record DL computations now handle (sparse) matrices with several millions of rows and columns.

9.3.4 The Number Field Sieve (NFS)

Many improvements for computing discrete logarithms first concerned prime fields and were adapted from improvements on integer factorization methods. Until 1993, the state-of-the-art algorithm for computing discrete logarithms in prime fields was the Coppersmith, Odlyzko, and Schroeppel (COS) algorithm [42] in $L_p[1/2, 1]$. In some cases, integer factorization was easier because the modulus had a special form. The equivalent for DL computation in prime fields is when p has a special form, given by a polynomial P of very small coefficients, $p = 2^{127} - 1$ for example. In that case, one can define an algebraic side with this polynomial P. By construction, P will have a root m modulo p of size $\sim p^{1/\deg P}$, hence the polynomial of the rational side ($g = U - Vx$ in the COS algorithm) will have coefficients of size $m \sim p^{1/\deg P}$. However, a generic method was found to reduce the coefficient size of the polynomials when one of the degrees increases. In 1993, Gordon [55] proposed the first version of NFS–DL algorithm for prime fields $\mathbb{F}_p$ with asymptotic complexity $L_p[1/3, 9^{1/3}]$. Gordon's $L_p[1/3]$ algorithm is interesting for very large values of p that were not yet targets for discrete logarithm computations in the 1990s. Buhler, H. Lenstra, and Pomerance [33] estimated the crossover point between 100 and 150 decimal digits, i.e., between 330 and 500 bits.

In Gordon's algorithm, Phase 1 and Phase 3 are modified. The Phase 2 is still a large sparse matrix kernel computation. We explain the polynomial selection method and the sieving phase. We also explain why the Phase 3 (individual logarithm computation) needs important modifications. The hurried reader can skip the proof of Theorem 9.12.

Polynomial selection with the base-m method

The polynomial selection of [55] is an analogy for prime fields of the method [33] for integer factorization. We will build a polynomial f of degree $d > 1$ and a polynomial g of degree 1 such that they have a common root m modulo p, and have coefficients of size $\sim p^{1/d}$. Set $m = [p^{1/d}]$ and write p to the base-m:

$$p = c_d m^d + c_{d-1} m^{d-1} + \ldots + c_0,$$

where $0 \leq c_i < m$. Then set

$$f = c_d x^d + c_{d-1} x^{d-1} + \ldots + c_0 \quad \text{and} \quad g = x - m \ .$$

Under the condition $p > 2^{d^2}$, f will be monic [33, Prop. 3.2]. These two polynomials have a common root m modulo p hence a similar map than in Equation (9.4) is available:

$$\begin{array}{rrcl} \rho: & \mathbb{Z}[x]/(f(x)) = \mathbb{Z}[\alpha_f] & \to & \mathbb{F}_p \\ & \alpha_f & \mapsto & m \bmod p \\ \text{hence} & a - b\alpha_f & \mapsto & a - bm \bmod p \ . \end{array} \tag{9.7}$$

Relation collection

The new technicalities concern factorization of ideals $a - b\alpha_f$ into prime ideals of $\mathbb{Z}[\alpha_f]$. This is not as simple as for $\mathbb{Z}[i]$: $\mathbb{Z}[\alpha_f]$ may not be a unique factorization domain, moreover what is called *bad ideals* can appear in the factorization. To end up with good relations, one stores only pairs (a, b) such that $a - b\alpha_f$ factors into *good* prime ideals of degree one, and whose norm is bounded by B. The sieve on the rational side is as in the COS algorithm.

Individual discrete logarithm computation

The other new issue is the individual logarithm computation. Since the sieving space and the factor basis $\mathcal{B}$ are much smaller ($L_p[1/3, \beta]$ instead of $L_p[1/2, \beta]$), there are much fewer known logarithms. It is hopeless to compute $g^t h$ with t at random until a smooth factorization is found, because now the smoothness bound is too small. A strategy in two phases was proposed by Joux and Lercier: Fix a larger smoothness bound $B_1 = L_p[2/3, \beta_1]$. First find a B_1-smoothness decomposition of $g^t h$. Secondly, treat separately each prime factor less than B_1 but larger than B to obtain a B-smooth decomposition. Finally retrieve the individual logarithm of h in base g. Each step has a cost $L_Q[1/3, c']$ with c' smaller than the constant $c = 1.923$ of the two dominating steps (relation collection and linear algebra).

Asymptotic complexity

We present how to obtain the expected heuristic running-time of $L_p[1/3, (\frac{64}{9})^{1/3}]$ to compute DL in $\mathbb{F}_p$ with the base-m method and a few improvements to the original Gordon algorithm. The impatient reader can admit the result of Theorem 9.12 and skip this section.

Theorem 9.12. *The running-time of the NFS-DL algorithm with base-m method is*

$$L_p\left[1/3, \left(\frac{64}{9}\right)^{1/3} \approx 1.923\right],$$

Table 9.3: Optimal value $\delta(\frac{\log p}{\log\log p})^{1/3}$ with $\delta = 1.44$ for p of 100 to 300 decimal digits. One takes $d = \lceil x \rceil$ or $d = \lfloor x \rfloor$ in practice.

$\log_{10} p$	40	60	80	100	120	140	160	180	200	220	240	260	280	300
$\lceil \log_2 B \rceil$ (bits)	18b	21b	24b	27b	29b	31b	33b	35b	36b	38b	39b	41b	42b	43b
$\delta \left(\frac{\log p}{\log\log p}\right)^{1/3}$	3.93	4.37	4.72	5.02	5.27	5.50	5.71	5.90	6.08	6.24	6.39	6.54	6.68	6.81

obtained for a smoothness bound $B = L_p[1/3, \beta]$ with $\beta = (8/9)^{1/3} \approx 0.96$, a sieving bound $E = B$ (s.t. $|a|, |b| < E$), and a degree of f to be $d = \lceil \delta(\frac{\log p}{\log\log p})^{1/3} \rfloor$ with $\delta = 3^{1/3} = 1.44$.

We present in Table 9.3 the optimal values of B (in bits) and d with $\beta \approx 0.96$ and $\delta \approx 1.44$ for p from 100 to 300 decimal digits (dd).

Proof. (of Theorem 9.12.) One of the key-ingredients is to set an optimal degree d for f. So let

$$d = \delta \left(\frac{\log p}{\log\log p}\right)^{1/3} \text{ so that } m = p^{1/d} = L_p\left[2/3, \frac{1}{\delta}\right] . \tag{9.8}$$

(Compute $\log m = \frac{1}{d}\log p = \frac{1}{\delta}\left(\frac{\log\log p}{\log p}\right)^{1/3} \log p = \frac{1}{\delta}\log^{2/3} p \log^{1/3}\log p$). We will compute the optimal value of δ under the given constraints later. The aim is to get a bound on the norms of the elements $a - b\alpha_f$ and $a - bm$ of size $L_p[2/3, \cdot]$ and a smoothness bound $L_p[1/3, \cdot]$, so that the smoothness probability will be $L_p[1/3, \cdot]$. A smoothness test is done with the Elliptic Curve Method (ECM). The cost of the test depends on the smoothness bound B and the total size of the integer tested. We first show that the cost of an ECM B-smoothness test with $B = L_p[1/3, \beta]$, of an integer of size $L_p[2/3, \eta]$ is $L_p[1/6, \sqrt{2\beta/3}]$, hence is negligible compared to any $L_p[1/3, \cdot]$. The cost of this ECM test depends on the size of the smoothness bound:

$$\text{cost of an ECM test} = L_B[1/2, \sqrt{2}] .$$

Writing $\log B = \beta \log^{1/3} p \log^{2/3}\log p$, we compute $\log L_B[1/2, \sqrt{2}] = \sqrt{2}(\log B \log\log B)^{1/2}$, cancel the negligible terms, and get the result.

We denote the infinity norm of a polynomial to be the largest coefficient in absolute value:

$$\|f\|_\infty = \max_{0 \le i \le \deg f} |f_i| . \tag{9.9}$$

We have

$$\|f\|_\infty, \|g\|_\infty \le m = L_p\left[\frac{2}{3}, \frac{1}{\delta}\right] .$$

In Phase 1, we sieve over pairs (a, b) satisfying $0 < a < E, -E < b < E$ and $\gcd(a, b) = 1$, so the sieving space is of order E^2. We know that we can sieve over no more than $L_p[1/3, \cdot]$ pairs to be able to balance the three phases of the algorithm. So let $E = L_p[1/3, \epsilon]$ pairs, with ϵ to be optimized later. The sieving space is $E^2 = L_p[1/3, 2\epsilon]$. Since the cost of a B-smoothness test with ECM is negligible compared to $L_p[1/3, \epsilon]$, we conclude that the running time of the sieving phase is $E^2 = L_p[1/3, 2\epsilon]$.

We need at least B relations to get a square matrix, and the linear algebra cost will be $B^2 = L_p[1/3, 2\beta]$. To balance the cost of the sieving phase and the linear algebra phase, we set

$$E^2 = B^2, \text{ hence } \epsilon = \beta$$

and we replace ϵ in the following computations.

What is the norm bound for $a - b\alpha_f$? We need it to estimate its probability to be B-smooth. The norm is computed as the *resultant* (denoted Res) of the element $a - b\alpha_f$ as a polynomial $a - bx$ in x, and the polynomial f. Then we bound the norm. The norm is

$$\begin{aligned}\mathcal{N}(a - b\alpha_f) &= \mathrm{Res}(f(x), a - bx) \\ &= a^d + f_{d-1}a^{d-1}b + \ldots + ab^{d-1}f_1 + b^d f_0 = \textstyle\sum_{i=0}^{d} a^i b^{d-i} f_i \\ &\leq (\deg f + 1)\|f\|_\infty E^{\deg f} = (d+1)p^{1/d}E^d\end{aligned}$$

with $d+1$ negligible, $p^{1/d} = m = L_p[2/3, 1/\delta]$, and $E^d = B^d = L_p[2/3, \beta\delta]$ (first compute $d \log B$ to get the result).

$$\mathcal{N}(a - b\alpha_f) \leq L_p[2/3, 1/\delta + \delta\beta] \ .$$

Then for the g-side we compute

$$\begin{aligned}a - bm &\leq Ep^{1/d} = L_p[1/3, \beta]L_p[2/3, \tfrac{1}{\delta}] \\ &\leq L_p[2/3, \tfrac{1}{\delta}]\end{aligned}$$

since the $L_p[1/3, \beta]$ term is negligible.

We make the usual heuristic assumption that the norm of $a - b\alpha_f$ follows the same smoothness probability as a random integer of the same size. Moreover, we assume that the probability of the norm of $a - b\alpha_f$ and $a - bm$ to be B-smooth at the same time is the same as the probability of their product (bounded by $L_p[2/3, 2/\delta + \delta\beta]$) to be B-smooth.

Finally we apply Corollary 9.11 and get

$$\Pr[\mathcal{N}(a - b\alpha_f) \text{ and } a - bm \text{ are } B\text{-smooth}] = 1/L_p\left[\frac{1}{3}, \frac{1}{3}\left(\frac{2}{\delta\beta} + \delta\right)\right].$$

How many relations do we have ? We multiply this smoothness probability Pr by the sieving space E^2 and obtain

$$\text{number of pairs } (a,b) \text{ tested} = \frac{\text{number of relations}}{B\text{-smoothness probability}} = \frac{B}{\Pr} = L_p\left[1/3, \beta + \frac{1}{3}\left(\frac{2}{\delta\beta} + \delta\right)\right] .$$

Since B^2 pairs were tested, we obtain the equation

$$2\beta = \beta + \frac{1}{3}\left(\frac{2}{\delta\beta} + \delta\right) \Leftrightarrow \beta = \frac{1}{3}\left(\frac{2}{\delta\beta} + \delta\right) \ . \tag{9.10}$$

We want to minimize the linear algebra and sieving phases, hence we minimize $\beta > 0$ through finding the minimum of the function $x \mapsto \frac{1}{3}(\frac{2}{\beta x} + x)$ (by computing its derivative): This is $2/3\sqrt{2/\beta}$, obtained with $\delta = x = \sqrt{2/\beta}$. We end up by solving Equation (9.10): $\beta = 2/3\sqrt{2/\beta} \Leftrightarrow \beta = (8/9)^{1/3}$. Since the running time of Phase 1 and Phase 2 is $L_p[1/3, 2\beta]$, we obtain $2\beta = (64/9)^{1/3}$ as expected. The optimal degree of the polynomial f is $d = \delta(\frac{\log p}{\log\log p})^{1/3}$ with $\delta = 3^{1/3} \approx 1.44$.

□

9.3.5 Number Field Sieve: Refinements

Norm approximation

We can approximate the norm of an element b in a number field $K_f = \mathbb{Q}[x]/(f(x))$ by the two following bounds.

The Kalkbrener bound [71, Corollary 2] is the following:

$$|\mathrm{Res}(f, b)| \leq \kappa(\deg f, \deg b) \cdot \|f\|_\infty^{\deg b}\|b\|_\infty^{\deg f}, \tag{9.11}$$

where $\kappa(n,m) = \binom{n+m}{n}\binom{n+m-1}{n}$, and $\|f\|_\infty = \max_{0\le i\le \deg f} |f_i|$ is the absolute value of the greatest coefficient. An upper bound for $\kappa(n,m)$ is $(n+m)!$.

Bistritz and Lifshitz proved the other following bound [25, Theorem 7]:

$$|\operatorname{Res}(f,\phi)| \le \|f\|_2^n \|\phi\|_2^m \le (m+1)^{n/2}(n+1)^{m/2}\|f\|_\infty^n\|\phi\|_\infty^m \ . \tag{9.12}$$

When the degree of the involved polynomials is negligible, we can approximate $|\operatorname{Res}(f,\phi)|$ by $O(\|f\|_\infty^n\|\phi\|_\infty^m)$. This simpler bound will be used to bound the norm of elements $\phi = a - bx$ and $\phi = \sum_{i=0}^{t-1} a_i x^i$ in a number field defined by a polynomial f.

Rational reconstruction and LLL

Throughout the polynomial selections, two algorithms are extensively used: the Rational Reconstruction algorithm and the Lenstra/Lenstra/Lovasz (LLL) algorithm. Given an integer y and a prime p, the Rational Reconstruction algorithm computes a quotient u/v such that $u/v \equiv y \mod p$ and $|u|, |v| < p$.

The Lenstra–Lenstra–Lovász algorithm (LLL) [83] computes a short vector in a lattice. Given a lattice $\mathcal{L}$ of $\mathbb{Z}^n$ defined by a basis given in an $n \times n$ matrix L, and parameters $\frac{1}{4} < \delta < 1$, $\frac{1}{2} < \eta < \sqrt{\delta}$, the LLL algorithm outputs a (η,δ)-*reduced basis* of the lattice. The coefficients of the first (shortest) vector are bounded by

$$(\delta - \eta^2)^{\frac{n-1}{4}} \det(L)^{1/n} \ .$$

With (η,δ) close to $(0.5, 0.999)$ (as in NTL or Magma), the approximation factor $C = (\delta-\eta^2)^{\frac{n-1}{4}}$ is bounded by 1.075^{n-1} (see [35, §2.4.2])). A very fast software implementation of the LLL algorithm is available with the `fplll` library [10].

Improvements of the polynomials

Joux and Lercier proposed in [65] another polynomial selection method that is an improvement of the base-m method. In this case, the polynomials are no longer monic but have smaller coefficients, bounded by $p^{\frac{1}{d+1}}$ instead of $p^{\frac{1}{d}}$. The size of the coefficients is spread over one more coefficient (the leading coefficient). Comeine and Semaev analyzed the complexity of their algorithm in [38]. The asymptotic complexity does not change because the gain is hidden in the $o(1)$ term. Their improvement is very important in practice, however.

For a given pair of polynomials (f,g) obtained with the Joux-Lercier method, one can improve their quality. We want the polynomials to have as many roots as possible modulo small primes, in order to get many relations in the relation collection step. The quality of the polynomials was studied my B. A. Murphy [93, 92]. The α value and the Murphy's $\mathbb{E}$ value measure the root properties of a pair of polynomials. The aim is to improve the root properties while keeping the coefficients of reasonable size. A recent work on this subject can be found in Shi Bai's PhD thesis [11] and in [12]. The sieving can be speeded-up in practice by choosing coefficients of the polynomial f whose size increases while the monomial degree decreases, and redefining the sieving space as $-E\sqrt{s} < a < E\sqrt{s}$, $0 < b < E/\sqrt{s}$ [74, 75].

9.3.6 Large Characteristic Non-Prime Fields

In 2006, Joux, Lercier, Smart, and Vercauteren [67] provided a polynomial construction that permitted us to conclude that for any finite field $\mathbb{F}_q$, there exists a $L_q[1/3, c]$ algorithm to compute DL: They proposed a method for medium-characteristic finite fields (known as the JLSV_1 method) and for large-characteristic finite fields. Another method (gJL: generalization of Joux-Lercier for prime fields) was independently proposed by Matyukhin [85] and Barbulescu [13, §8.3] and achieved the same asymptotic complexity as for prime

fields: $L_Q[1/3, (\frac{64}{9})^{1/3}]$, with $Q = p^n$. The two polynomials are of degree $d+1$ and $d \geq n$, for a parameter d that depends on $\log Q$ as for the prime case. Note that the optimal choice of d for the gJL method is $d = \delta \left(\frac{\log p}{\log\log p}\right)^{1/3}$ with $\delta = 3^{1/3}/2 \approx 0.72$ instead of $\delta = 3^{1/3}$ for the NFS-DL algorithm in prime fields. This is not surprising: In this case the sum of polynomial degrees $\deg f + \deg g$ is $2d+1$ instead of $d+1$.

Recent results (2015) on DL record computation in non-prime finite fields with the NFS algorithm

In 2015, Barbulescu, Gaudry, Guillevic, and Morain published a DL record computation in a 595-bit quadratic extension $\mathbb{F}_{p^2}$, where p is a generic 298-bit prime [16]. They designed a new polynomial selection method called the Conjugation. The polynomials allowed a factor-two speed-up in the relation collection step, thanks to an order-two Galois automorphism.

The area is moving and later in 2015, Sarkar and Singh in [100] proposed a variant that combines the gJL and the Conjugation method. The asymptotic complexity is the same: $L_Q[1/3, (\frac{64}{9})^{1/3}]$ and the polynomials might provide slightly smaller norm values in practice.

We lack one last piece of information to be able to recommend parameter sizes for a given level of security in large characteristic fields $\mathbb{F}_Q$: record computations with one of these methods.

9.3.7 Medium Characteristic Fields

This range of finite fields is where the discrete logarithm is more difficult to compute at the moment. Moreover, the records published are for quite small sizes of finite fields only: $\mathbb{F}_{p^3}$ of 120dd (400 bits) in [67], and in [121].

There was a big issue in obtaining an $L[1/3]$ algorithm as for the FFS algorithm in small characteristic and the NFS algorithm in large characteristic. The solution proposed in [67] introduces a modification in the relation collection. The small elements are of higher degree $t-1 \geq 2$, where t is of the form $\frac{n}{t} = \frac{1}{c_t}\left(\frac{\log Q}{\log\log Q}\right)^{1/3}$. To obtain similar asymptotic formulas in $L_Q[1/3, \cdot]$ as for prime fields (see Section 9.3.4), one sets the total number of elements considered in the relation collection to be E^2. With these settings,

- the total number of elements $\phi = \sum_{i=0}^{t-1} a_i x^i$ considered in the relation collection is $\|\phi\|_\infty^t = E^2$, so that $\|\phi\|_\infty = E^{2/t}$;
- the norm of the elements ϕ on the f side is

$$|\mathcal{N}_f(\phi)| \leq \mathrm{Res}(\phi, f) \leq \kappa(t-1, \deg f)\|\phi\|_\infty^{\deg f}\|f\|_\infty^{t-1} = \kappa(t-1, \deg f)E^{2\deg f/t}\|f\|_\infty^{t-1}.$$

The two polynomials defined for the 120dd record computation in [67] were $f = x^3 + x^2 - 2x - 1$ and $g = f + p$. This method is not designed for scaling well and the authors propose a variant where the two polynomials have a balanced coefficient size of $p^{1/2}$ each. This polynomial selection method, combined with the relation collection over elements of degree $t-1$, provides an asymptotic complexity of $L_Q[1/3, (\frac{128}{9})^{1/3} \simeq 2.42]$. This asymptotic complexity went down in 2015 in [16] to $L_Q[1/3, (\frac{96}{9})^{1/3} \simeq 2.201]$. These two asymptotic complexities were improved in [19, 96] to $L_Q[1/3, 2.39]$ and $L_Q[1/3, 2.156]$, respectively, by using a *multiple number field sieve* variant that we explain in the next paragraph. This variant has not been implemented for any finite field yet.

Multiple number field sieve

This paragraph is about refinements in the algorithm that slightly improve the asymptotic complexity but whose practical gain is not known, and not certain yet.

The idea is to use additional number fields and hope to generate more relations per polynomial (the $a - bx$ elements in Gordon's algorithm, for example). There are again two versions: one asymmetric where one number field is preferred, say f_0, and additional number fields f_i are considered. For each element $a - bx$, one tests the smoothness of the image of $a - bx$ first in the number field defined by f_0, and if successful, then in all of the number fields defined by the f_i, to generate relations. This is used with a polynomial selection that produces the first polynomial much better than the second. This is the case for the base-m method and was studied by Coppersmith [41]. The same machinery applies to the generalized Joux-Lercier method [85, 96]. In these two cases, the asymptotic complexity is $L_Q[1/3, 1.90]$ instead of $L_Q[1/3, 1.923]$ (where $Q = p^n$). This MNFS version also applies to the Conjugation method [96] and the complexity is $L_Q[1/3, 2.156]$ instead of $L_Q[1/3, 2.201]$.

The second symmetric version tests the smoothness of the image of $a - bx$ in all the pairs of possible number fields defined by f_i, f_j (for $i < j$). It applies to the NFS algorithm used with the JLSV_1 polynomial selection method, in medium characteristic. The complexity is $L_Q[1/3, 2.39]$ instead of $L_Q[1/3, 2.42]$.

Unfortunately, none of these methods were ever implemented (yet), even for a reasonable finite field size, hence we cannot realize how much in practice the smaller c constant improves the running-time. There was a similar unknown in 1993. The cross-over point between the Coppersmith-Odlyzko-Schroeppel algorithm in $L_p[1/2]$ and Gordon's algorithm in $L_p[1/3]$ was estimated in 1995 at about 150 decimal digits. In the MNFS algorithm, the constant is slightly reduced but no one knows the size of Q for which "in practice", a MNFS variant will be faster than a regular NFS algorithm.

Special-NFS, Tower-NFS, and pairing-friendly families of curves

This paragraph lists the improvements to the NFS algorithm dedicated to fields $\mathbb{F}_{p^n}$ where the prime p is of special form, i.e., given by a polynomial evaluated at a given value. This is the case for embedding fields of pairing-friendly curves in families, such as the Barreto-Naehrig curves.

When the prime p defining the finite field is of a special form, there exist better algorithms, such as the Special NFS (SNFS) for prime fields. Joux and Pierrot [69] proved a complexity of

$$L_Q\left[\frac{1}{3}, \left(\frac{\deg P + 1}{\deg P}\frac{64}{9}\right)^{1/3}\right]$$

with P the polynomial defining the prime p. We have $\deg P = 4$, for example, for a BN curve [21]. Note that when $\deg P = 2$ as for MNT curves, the complexity $L_Q[1/3, (\frac{96}{9})^{1/3}]$ is the same as the conjugation method complexity. This promising method has not yet been implemented.

Very recently, [18] proposed another construction named the Tower-NFS that would be the best choice for finite field target groups, where $\deg P \geq 4$. One of the numerous difficult technicalities of this variant is the degree of the polynomials: a multiple of n. Handling such polynomials and elements of high degree throughout the algorithm, in particular in the relation collection phase, is a very difficult task, and is not implemented at the moment.

A clear rigorous recommendation of parameter sizes is not possible at the moment for pairing target groups, since the area is not fixed. Theoretical algorithms are published but real-life implementations and records over reasonable-size finite fields (more than 512 bits) to estimate their running-time are not available.

We can clearly say that all these theoretical propositions shall be seriously taken into consideration. The constant c in the $L_Q[1/3, c]$ asymptotic formula is decreasing, and might reach the $(\frac{64}{9})^{1/3}$ value of prime fields one day. A generic recommendation in the large characteristic case, based on a $L_Q[1/3, (64/9)^{1/3}]$ asymptotic complexity, seems reasonable.

9.3.8 Small Characteristic: From the Function Field Sieve (FFS) to the Quasi-Polynomial-Time Algorithm (QPA)

The main difference between a small characteristic and a large-characteristic field is the Frobenius map $\pi : x \mapsto x^p$, where p is the characteristic of the field. This map is intensively used to obtain multiple relations from an initial one, at a negligible cost. For prime fields, the Frobenius map is the identity, so we cannot gain anything with it. In small-characteristic fields, the use of the Frobenius map is one of the key ingredients that provides a much better asymptotic complexity. This should be combined with a specific field representation that allows a very fast evaluation of the Frobenius map.

For large-characteristic finite fields $\mathbb{F}_{p^n}$, the Frobenius map of order n can provide a speed-up of a factor up to n if the polynomial selection provides a compatible representation, i.e., if a is B-smooth, then we want $\pi_p(a)$ to be B-smooth as well.

Example 9.13 (Systematic equations in $\mathbb{F}_{2^{127}}$)**.** Blake, Fuji-Hara, Mullin, and Vanstone in [27] targeted the finite field $\mathbb{F}_{2^{127}}$. They used the representation $\mathbb{F}_{2^{127}} = \mathbb{F}_2[x]/(x^{127}+x+1)$ to implement Adleman's algorithm. The polynomial $f(x) = x^{127}+x+1$ is primitive so that they have chosen x as a generator of $\mathbb{F}^*_{2^{127}}$. They observed that $x^{2^7-1} = x^{127} = x + 1 \bmod f(x)$, $x^{2^7} \equiv x^2 + x \bmod f(x)$ and $x^{2^i} \equiv x^{2^{i-6}} + x^{2^{i-7}} = x^{2^{i-7}}(1 + x^{2^{i-7}}) \bmod f(x)$ for any $i \geq 7$. Moreover, $(1 + x^{2^i}) = (1 + x)^{2^i} \equiv (x^{127})^{2^i}$ so that $\log_x(1 + x^{2^i}) = 127 \cdot 2^i$. Combining these equations, they obtain logarithms for free.

A second notable difference between small-characteristic fields and prime fields is the cost of factorization. In Algorithm 9.10, a preimage in $\mathbb{N}$ of elements in $\mathbb{F}_p$ is factorized. In small characteristic, elements of $\mathbb{F}_{2^n}$ are lifted to polynomials of $\mathbb{F}_2[x]$, then factorized. Over finite fields, there exists polynomial time algorithms to factor polynomials, hence the time needed per smooth test is much lower.

Brief history

Figure 9.4 shows the records in small-characteristic finite fields with the Function Field Sieve (FFS) and its various improvements, especially from 2012. Most of the records were announced on the number theory list [2]. Finite fields of characteristic 2 and 3 and composite extension degree such as target groups of pairing-friendly (hyper-)elliptic curves must definitively be avoided, since fields of even more than 3072 bits were already reached in 2014.

The Waterloo algorithm In 1984, Blake, Fuji-Hara, Mullin, and Vanstone proposed a dedicated implementation of Adleman's algorithm to $\mathbb{F}_{2^{127}}$ [27, 28]. They introduced the idea of *systematic equations* (using the Frobenius map) and *initial splitting* (this name was introduced later). Their idea works for finite fields of characteristic two and extension degree close to a power of 2 (e.g., 127). The asymptotic complexity of their method was $L_Q[1/2]$. In the same year, Blake, Mullin, and Vanstone [28] proposed an improved algorithm known as the Waterloo algorithm. Odlyzko computed the asymptotic complexity of this algorithm in [94]. The asymptotic complexity needs the estimation of the probability that a random polynomial over $\mathbb{F}_q$ of degree m factors entirely into polynomials of degree at most b, i.e., is b-smooth. Odlyzko in [94, Equation (4.5), p. 14] gave the following estimation.

$$p(m, n) = \exp\left((1 + o(1))\frac{n}{m}\log_e \frac{m}{n}\right) \text{ for } n^{1/100} \leq m \leq n^{99/100} . \tag{9.13}$$

The *initial splitting* idea is still used nowadays, combined with the QPA algorithm. Given a random element $a(x)$ of GF(2^n) represented by a degree $n-1$ polynomial over GF(2)

[2] `https://listserv.nodak.edu/cgi-bin/wa.exe?A0=NMBRTHRY`

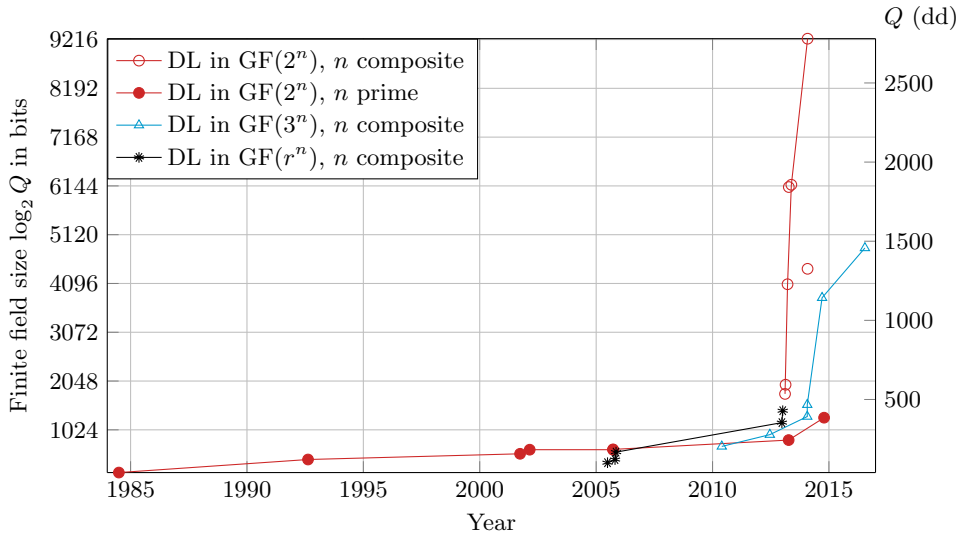


Figure 9.4: Records of DL computation in fields $\mathbb{F}_{2^n}, \mathbb{F}_{3^n}, \mathbb{F}_{r^n}$ of small characteristic, with n prime or composite. All the fields $\mathbb{F}_{2^n}, \mathbb{F}_{3^n}$ with n composite are target fields of supersingular pairing-friendly (hyper-)elliptic curves.

modulo an irreducible degree n polynomial $f(x)$, the algorithm computes the extended Euclidean algorithm to compute the GCD of $f(x)$ and $a(x)$. At each iteration, the following equation holds [27, §2]:

$$s_i(x)a(x) + t_i(x)f(x) = r_i(x) \ . \tag{9.14}$$

Reducing this equation modulo $f(x)$, one obtains $a(x) \equiv r_i(x)/s_i(x) \bmod f(x)$. The degree of $r_i(x)$ decreases while the degree of $s_i(x)$ increases. By stopping the extended Euclidean algorithm at the state i where $\deg r_i(x), \deg s_i(x) \leq \lfloor n/2 \rfloor$, one obtains the initial splitting of $a(x)$ of degree $n-1$ into two polynomials $r_i(x), s_i(x)$ of degree at most $\lfloor n/2 \rfloor$.

Odlyzko computed the asymptotic complexity of the Waterloo algorithm to be [94, Equation (4.17), p. 19] $L_Q[1/2, (2\log_e(2))^{1/2} \approx 1.1774]$.

Coppersmith's $L_Q[1/3]$ algorithm and FFS algorithm Building on the idea of systematic equations, Coppersmith [40] gave the first $L_Q[1/3, c]$ algorithm for DL computations over $\mathbb{F}_{2^n}$ (with $Q = 2^n$). He found $(32/9)^{1/3} \leq c \leq 4^{1/3}$ and did a record computation for $\mathbb{F}^*_{2^{127}}$. In 1994, Adleman [7] generalized this work to the case of any small characteristic, and this is now called the Function Field Sieve (FFS). This gave a $L_Q[1/3, (\frac{64}{9})^{\frac{1}{3}}]$ for Q of small characteristic, with function field (in place of number field for prime fields). Later, Adleman-Huang improved that to $L_Q[1/3, (\frac{32}{9})^{\frac{1}{3}}]$ for Q of small characteristic [8], however, this was slower than Coppersmith for $\mathbb{F}_{2^n}$.

Outside of the pairing-based cryptography context, the research and the records are focused on prime extensions degrees. In 2002 Thomé increased the Coppersmith record up to GF(2^{607}) [115, 116]. During the same time, Joux and Lercier implemented FFS for GF(2^{521}) in [64]. Continuing the record series, in 2005, Joux and Lercier recomputed a record in GF(2^{607}) and went slightly further with a record in GF(2^{613}). They also investigated the use of FFS for larger characteristic finite fields in [66]. In 2013, a record of the CARAMEL group in GF(2^{089}) with FFS was announced by Bouvier on the NMBRTHRY list [14] and published in [15]. The actual record is held by Kleinjung, in GF(2^{1279}) [76].

Since 2000, examples of supersingular pairing-friendly elliptic curves of cryptographic size arise. Two curves are well studied in characteristic 2 and 3 for the various speed-up they

provide, in particular, in hardware. Supersingular curves of almost prime order in small characteristic are very rare. No ordinary pairing-friendly curves were ever known in small characteristic. The embedding degree for supersingular curves is at most 4 in characteristic 2 and at most 6 in characteristic 3. The first cryptanalists exploited this composite degree extension to improve the FFS algorithm. In 2010, the record in characteristic three was in $\mathrm{GF}(3^{6\cdot 71})$ of 676 bits [59]. In 2012, due to increasing computer power and the prequel of the use of the additional structure provided by the composite extension degree of the finite field, a DL record-breaking in $\mathrm{GF}(3^{6\cdot 97})$ (a 923-bit finite field) was made possible [47]. This announcement had a quite important effect over the community at that time, probably because the broken curve was the one used in the initial paper on short signatures from pairings [29]. The targeted finite field $\mathbb{F}_{3^{582}}$ was the target field of a pairing-friendly elliptic curve in characteristic 3, considered safe for 80-bit security implementations.

The real start of mathematical improvements occurred at Christmas 2012: Joux proposed a conjectured heuristic $L_Q[1/4]$ algorithm [63] and announced two records [62] of much larger size. In finite fields that can be represented as Kummer extensions, using the Frobenius gives many relations at one time, hence speeding-up the relation-collection phase. As we can see in Figure 9.4, records in prime extensions n of $\mathbb{F}_2$ do not grow as extraordinary as composite extensions coming from pairing-friendly curves.

The Quasi-Polynomial-time Algorithm (QPA) In 2013 [48] an improved *descent* phase was proposed, that provided a quasi-polynomial-time algorithm (QPA). Two variants of the algorithm were published [17, 56]. The polynomial selection differs and induces differences in the algorithm. We are at the "beginning of the end"— much work is still needed for a complete implementation of this algorithm. In particular, the descent phase is still costly in memory requirements.

The two versions of the QPA algorithm intensively exploit the Frobenius map, to obtain many relations for free. It works when the extension degree n is composite and satisfies some properties.

As a conclusion, we list the last records published. In 2014 Adj, Menezes, Oliveira, and Rodrígues-Henríquez published a discrete logarithm record in $\mathrm{GF}(3^{6\cdot 137})$ and $\mathrm{GF}(3^{6\cdot 163})$, corresponding to a 1303-bit and a 1551-bit finite field [5]. In 2014, Joux and Pierrot published a record in $\mathrm{GF}(3^{5\cdot 479})$ corresponding to a 3796-bit field [68]. In 2014, Granger, Kleinjung, and Zumbragel announced a record in $\mathrm{GF}(2^{9234})$ [57].

Recent improvements in finite fields of composite extension degree (Spring 2016) There were major theoretical improvements in finite fields $\mathbb{F}_{p^n}$ where n is composite, in 2015 and 2016. This paragraph tries to summarize the news. Two preprints by Kim on one side and Barbulescu on the other side evolved to a common paper at CRYPTO'16 [73]. In parallel, Sarkar and Singh combined Kim–Barbulescu's work with their own techniques [103, 101, 102]. We need to mention Jeong and Kim's work [61] to complete the list or recent preprints on the subject. This one paper and these four preprints exploit the extension degree that should be composite. They each propose an improved polynomial selection step that allows us to reduce the size of the norms of the elements involved in the relation collection. Since the improvement is notable, it reduces the asymptotic complexity of the NFS algorithm. These papers exploit the finite field structure and contruct a degree-n extension as a tower of three levels: a base field $\mathbb{F}_p$ as first level, a second level $\mathbb{F}_{p^\eta}$, and a third level $\mathbb{F}_{p^{\eta\kappa}} = \mathbb{F}_{p^n}$. The extension degree n should be composite and the divisor η of quite small size. The two (or multiple) number fields will exploit this structure as well. This setting provides the following new asymptotic complexities for medium-characteristic fields:

1. $L_{p^n}[1/3, (48/9)^{1/3} \approx 1.747]$ when n is composite, $n = \eta\kappa$, $\kappa = \left(\frac{1}{12^{1/3}} + o(1)\right)\left(\frac{\log Q}{\log\log Q}\right)^{1/3}$, and p is generic;

2. $L_{p^n}[1/3, (32/9)^{1/3} \approx 1.526]$ when n is composite and p has a special form.

The generic case where n is prime is not affected by these new improvements. We summarize in Table 9.4 the new theoretical security of a pairing-friendly curve where (1) n is composite and (2) n is composite and p of special form, for p^n of 3072 bits.

Table 9.4: Estimate of security levels according to NFS variants.

$\log_2 p^n$	Conj $L_{p^n}[\frac{1}{3}, 2.20]$	Joux–Pierrot $d=4$ $L_{p^n}[\frac{1}{3}, 2.07]$	– $L_{p^n}[\frac{1}{3}, 1.923]$	Conj Ext. TNFS $L_{p^n}[\frac{1}{3}, 1.747]$	Special Ext. TNFS $L_{p^n}[\frac{1}{3}, 1.526]$
3072	$2^{159-\delta_1}$	$2^{149-\delta_2}$	$2^{139-\delta_3}$	$2^{126-\delta_4}$	$2^{110-\delta_5}$
3584	$2^{169-\delta_1}$	$2^{159-\delta_2}$	$2^{148-\delta_3}$	$2^{134-\delta_4}$	$2^{117-\delta_5}$
4096	$2^{179-\delta_1}$	$2^{169-\delta_2}$	$2^{156-\delta_3}$	$2^{142-\delta_4}$	$2^{124-\delta_5}$
4608	$2^{188-\delta_1}$	$2^{177-\delta_2}$	$2^{164-\delta_3}$	$2^{149-\delta_4}$	$2^{130-\delta_5}$
5120	$2^{197-\delta_1}$	$2^{185-\delta_2}$	$2^{172-\delta_3}$	$2^{156-\delta_4}$	$2^{136-\delta_5}$
5632	$2^{204-\delta_1}$	$2^{192-\delta_2}$	$2^{179-\delta_3}$	$2^{162-\delta_4}$	$2^{142-\delta_5}$
6144	$2^{212-\delta_1}$	$2^{199-\delta_2}$	$2^{185-\delta_3}$	$2^{168-\delta_4}$	$2^{147-\delta_5}$

Note: The numbers should be read as follows: a 3072-bit finite field, which is the embedding field of a BN curve whose p is of special form and n is composite will provide approximately a security level of $2^{110-\delta_{\mathrm{BN}}}$, where δ_{BN} depends on the curve and on the implementation of the special extended NFS variant.

9.3.9 How to Choose Real-Size Finite Field Parameters

At some point, to design a cryptosystem, we want to translate an asymptotic complexity to a size recommendation for a given security level, usually equivalent to an AES level of security: 128, 192, or 256 bits. In other words, we would like that for a given finite field $\mathbb{F}_q$ of given size, the running-time required to break an instance of DLP is equivalent to 2^{128}, 2^{192}, or 2^{256} group operations. For the DLP in a generic group, we saw in Section 9.2 that the expected time is in $O(\sqrt{N})$, with N the prime-order subgroup considered. A group of size $2n$ bits (where only generic attacks apply) is enough to achieve an n-bit security level.

We present in Table 9.5 the usual key length recommendations from `http://www.keylength.com`. The NIST recommendations are the less-conservative ones. A modulus of length 3072 is recommended to achieve a security level equivalent to a 128-bit symmetric key. The ECRYPT II recommendations are slightly larger: 3248 bit modulus are suggested.

Table 9.5: Cryptographic key length recommendations, August 2015.

Method	Date	Sym-metric	Asymmetric	Discrete Log		Elliptic curve	Hash function
				Key	Group		
Lenstra / Verheul [84]	2076	129	6790–5888	230	6790	245	257
Lenstra Updated [82]	2090	128	4440–6974	256	4440	256	256
ECRYPT II (EU) [1]	2031–2040	128	3248	256	3248	256	256
NIST (US) [4]	> 2030	128	3072	256	3072	256	256
ANSSI (France) [3]	2021–2030	128	2048	200	2048	256	256
NSA (US) [2]	–	128	–	–	–	256	256
RFC3766 [95]	–	128	3253	256	3253	242	–

Note: All key sizes are provided in bits. These are the minimal sizes for security.

We explain here where these key sizes come from. The running-time complexity of the most efficient attacks on discrete logarithm computation and factorization are considered

and balanced to fit the last records. In practice, we calibrate the asymptotic complexity (we set the constant hidden in the $O()$ notation) so that it matches the largest DL record computations. For prime fields $\mathbb{F}_p$ with no special form of the prime p, the asymptotic formula of NFS-DL is $L_p[1/3, (\frac{64}{9})^{1/3}]$, and we consider its logarithm in base 2:

$$\log_2 L[\alpha, c](n) = \big(c + o(1)\big)\ n^{\alpha} \log_2^{1-\alpha}(n \ln 2) \tag{9.15}$$

with $n = \log_2 N$. The last record was a DL computation in a prime field of 180dd or 596 bits, `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1406&L=NMBRTHRY&F=&S=&P=3161`.

Figure 9.5 presents the records of DL computation in prime fields, the records of RSA modulus factorization and an interpolation according to [81, §3] by a Moore law doubling every nine months.

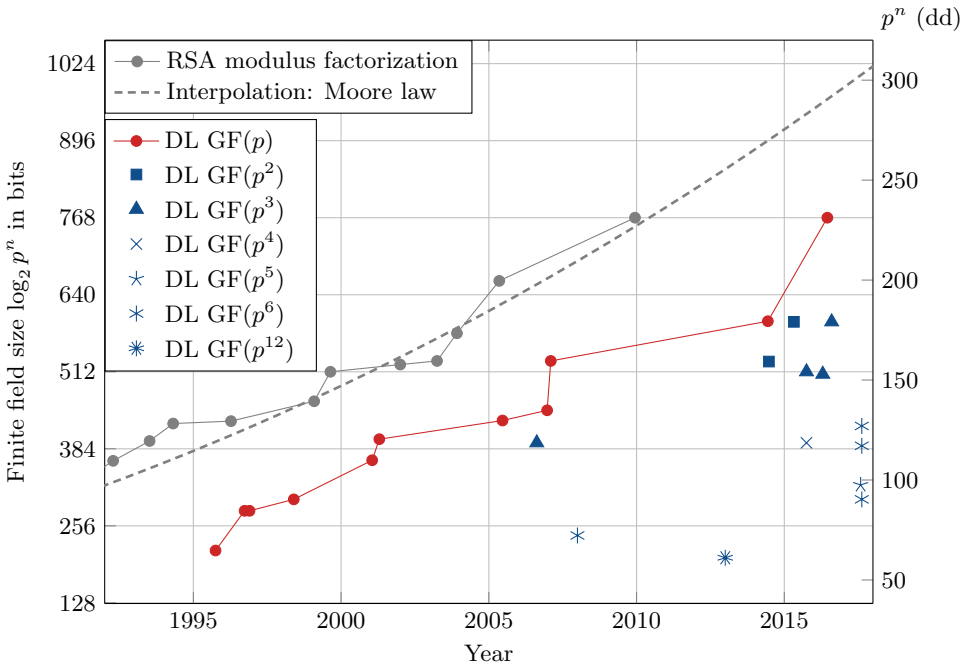


Figure 9.5: Records of DL computation in prime fields and RSA modulus factorization

To estimate the required modulus size, we compute the logarithm in base 2 of the L-notation (9.15) and translate it such that $\log_2 L[c, \alpha](598) \approx 60$ (with 180dd=598bits). We obtain $\log_2 L[c, \alpha](598) = 68.5$ so we set $a = -8.5$. We obtain $\log_2 L[c, \alpha](3072) - 8.5 = 130$ so we can safely deduce that a 3072-bit prime field with a generic safe prime is enough to provide a 128-bit security level.

Conservative recommendations. To avoid dedicated attacks, and specific NFS variants, common-sense advice would be to avoid the curves with too much structure in the parameters. Here is a list of points to take into account.

- Use a generic curve constructed with the Cocks-Pinch or Dupont-Enge Morain methods;
- Use a curve in a family with a non-special form seed, i.e., the prime $p = P(x_0)$ is such that x_0 has no special form (e.g., $x_0 \neq 2^{63} + 1$);

- Use a curve with low-degree polynomials defining the parameters, e.g., degree 2 (MNT and Galbraith-McKee-Valença curves) or degree 4 (Freeman curves);
- Use a curve whose discriminant D is large (e.g., constructed with the Cocks-Pinch or Dupont-Enge-Morain method, or an MNT, a Galbraith-McKee-Valença, or a Freeman curve);
- Use a prime embedding degree.

9.3.10 Discrete logarithm algorithms in pairing-friendly target finite fields $\mathbb{F}_{p^n}$: August 2016 state-of-the-art

Given a finite field $\mathbb{F}_{p^n}$ which contains the target group of a cryptographic pairing, different NFS-based algorithms can be applied to compute discrete logarithms, depending on the structure of the finite field. Two criterias should be taken into account: whether n is prime, and whether the characteristic p has a special form: given by a polynomial of degree greater than two.

1. If n is prime,
   (a) and p has no special form (e.g., supersingular curves where $k = 2$, MNT curves where $n = 3$, and any curves constructed with the Cocks-Pinch or Dupont-Enge-Morain methods), then only the generic NFS algorithms apply.
      i. In a large-characteristic finite field, the generalized Joux–Lercier method of asymptotic complexity $L_Q[1/3, 1.923]$ (and $L_Q[1/3, 1.90]$ in the multiple-NFS version) applies.
      ii. In a medium-characteristic finite field, the conjugation method of asymptotic complexity $L_Q[1/3, 2.20]$ applies. The multiple-NFS version has an asymptotic complexity of $L_Q[1/3, 2.15]$. The finite field size does not need to be enlarged for now.

   In practice for large sizes of finite fields, the Sarkar-Singh method that interpolates between the GJL and the Conjugation methods provides smaller norms. In this case, the key size should be enlarged by maybe 10% but not significantly since the asymptotic complexity is not lower than the complexity of NFS in a prime field: $L_Q[1/3, 1.923]$.
   (b) If p is given by a polynomial of degree at least three, i.e., $p = P(u)$ where $\deg(P) \geq 3$, then the Joux–Pierrot method applies. In the medium-characteristic case, the asymptotic complexity tends to $L_Q[1/3, 1.923]$ for large $\deg(P)$. In large characteristic, the pairing-friendly curves ($k = 2, 3, 4, 6$ for instance) are such that $\deg(P) = 2$ only.
2. If n is composite, then the extended tower-NFS technique, firstly introduced by Kim then improved by Barbulescu, Kim, Sarkar and Singh, and Jeong, applies.
   (a) If p has no special form or is given by a polynomial of degree at most 2 (MNT curves of embedding degree 4 and 6, Cocks-Pinch and Dupont-Enge-Morain methods), then the asymptotic complexity is $L_Q[1/3, 1.74]$ so asymptotically, the finite field size should be enlarged by a factor 4/3.
   (b) If p has a special form, then the asymptotic complexity is $L_Q[1/3, 1.56]$ and asymptotically, the finite field size should be doubled.

Bibliography

[1] D.SPA.20. *ECRYPT2 Yearly Report on Algorithms and Keysizes (2011-2012)*. European Network of Excellence in Cryptology II, September 2012.

[2] NSA Suite B. *Fact Sheet Suite B Cryptography*. National Security Agency, U.S.A., September 2014.

[3] RGS-B1. *Mécanismes cryptographiques - Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques*. Agence Nationale de la Sécurité des Systèmes d'Information, France, February 2014. version 2.03.

[4] SP-800-57. *Recommendation for Key Management – Part 1: General*. National Institute of Standards and Technology, U.S. Department of Commerce, July 2012.

[5] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ using Magma. In Ç. K. Koç, S. Mesnager, and E. Savas, editors, *Arithmetic of Finite Fields (WAIFI 2014)*, volume 9061 of *Lecture Notes in Computer Science*, pp. 3–22. Springer, 2014.

[6] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *20th Annual Symposium on Foundations of Computer Science*, pp. 55–60. IEEE Computer Society Press, 1979.

[7] Leonard Adleman. The function field sieve. In L. M. Adleman and M.-D. Huang, editors, *Algorithmic Number Theory (ANTS-I)*, volume 877 of *Lecture Notes in Computer Science*, pp. 141–154. Springer, 1994.

[8] Leonard M. Adleman and Ming-Deh A. Huang. Function field sieve method for discrete logarithms over finite fields. *Information and Computation*, 151(1/2):5–16, 1999.

[9] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In I. Ray, N. Li, and C. Kruegel, editors, *22nd ACM Conference on Computer and Communications Security*, pp. 5–17. ACM Press, 2015.

[10] M. Albrecht, S. Bai, D. Cadé, X. Pujol, and D. Stehlé. fplll-4.0, a floating-point LLL implementation. Available at `http://perso.ens-lyon.fr/damien.stehle`.

[11] Shi Bai. *Polynomial Selection for the Number Field Sieve*. PhD thesis, Australian National University, 2011. `http://maths.anu.edu.au/~brent/pd/Bai-thesis.pdf`.

[12] Shi Bai, Richard Brent, and Emmanuel Thomé. Root optimization of polynomials in the number field sieve. *Mathematics of Computation*, 84(295):2447–2457, 2015.

[13] Razvan Barbulescu. *Algorithmes de logarithmes discrets dans les corps finis*. PhD thesis, Université de Lorraine, 2013. `https://tel.archives-ouvertes.fr/tel-00925228`.

[14] Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau, and Paul Zimmermann. Discrete logarithm in GF(2^{809}) with ffs, April 2013. Announcement available at the NMBRTHRY archives, item 004534.

[15] Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau, and Paul Zimmermann. Discrete logarithm in GF(2^{809}) with FFS. In H. Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pp. 221–238. Springer, Heidelberg, 2014.

[16] Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain. Improving NFS for the discrete logarithm problem in non-prime finite fields. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pp. 129–155. Springer, Heidelberg, 2015.

[17] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pp. 1–16. Springer, Heidelberg, 2014.

[18] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. The tower number field sieve. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pp. 31–55. Springer, Heidelberg, 2015.

[19] Razvan Barbulescu and Cécile Pierrot. The Multiple Number Field Sieve for Medium and High Characteristic Finite Fields. *LMS Journal of Computation and Mathematics*, 17:230–246, 2014.

[20] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks (SCN 2002)*, volume 2576 of *Lecture Notes in Computer Science*, pp. 257–267. Springer, Heidelberg, 2003.

[21] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography (SAC 2005)*, volume 3897 of *Lecture Notes in Computer Science*, pp. 319–331. Springer, Heidelberg, 2006.

[22] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In S. D. Galbraith and M. Nandi, editors, *Progress in Cryptology – INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pp. 317–338. Springer, Heidelberg, 2012.

[23] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In K. Sako and P. Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pp. 321–340. Springer, Heidelberg, 2013.

[24] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. On the correct use of the negation map in the Pollard rho method. In D. Catalano et al., editors, *Public Key*

Cryptography – PKC 2011, volume 6571 of *Lecture Notes in Computer Science*, pp. 128–146. Springer, Heidelberg, 2011.

[25] Yuval Bistritz and Alexander Lifshitz. Bounds for resultants of univariate and bivariate polynomials. *Linear Algebra and its Applications*, 432(8):1995–2005, 2009.

[26] Simon R. Blackburn and Edlyn Teske. Baby-step giant-step algorithms for non-uniform distributions. In W. Bosma, editor, *Algorithmic Number Theory (ANTS-IV)*, volume 1838 of *Lecture Notes in Computer Science*, pp. 153–168. Springer, 2000.

[27] Ian F. Blake, Ryoh Fuji-Hara, Ronald C. Mullin, and Scott A. Vanstone. Computing logarithms in finite fields of characteristic two. *SIAM Journal on Algebraic Discrete Methods*, 5(2):276–285, 1984.

[28] Ian F. Blake, Ronald C. Mullin, and Scott A. Vanstone. Computing logarithms in $\mathrm{GF}(2^n)$. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pp. 73–82. Springer, Heidelberg, 1984.

[29] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pp. 514–532. Springer, Heidelberg, 2001.

[30] Cyril Bouvier. *Algorithmes pour la factorisation d'entiers et le calcul de logarithme discret*. PhD thesis, Université de Lorraine, 2015. `https://tel.archives-ouvertes.fr/tel-01167281`.

[31] Richard P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20:176–184, 1980.

[32] Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[33] Joe P. Buhler, Hendrik W. Lenstra Jr., and Carl Pomerance. Factoring integers with the number field sieve. In A. K. Lenstra and H. W. Lenstra Jr., editors, *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*, pp. 50–94. Springer, 1993.

[34] Earl R. Canfield, Paul Erdős, and Carl Pomerance. On a problem of Oppenheim concerning "factorisatio numerorum". *Journal of Number Theory*, 17(1):1–28, 1983.

[35] Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Université Paris 7 Denis Diderot, 2013. `http://www.di.ens.fr/~ychen/research/these.pdf`.

[36] Jung Hee Cheon, Jin Hong, and Minkyu Kim. Speeding up the pollard rho method on prime fields. In J. Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pp. 471–488. Springer, Heidelberg, 2008.

[37] Young Ju Choie, Eun Kyung Jeong, and Eun Jeong Lee. Supersingular hyperelliptic curves of genus 2 over finite fields. *Journal of Applied Mathematics and Computation*, 163(2):565–576, 2005.

[38] An Commeine and Igor Semaev. An algorithm to solve the discrete logarithm problem with the number field sieve. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pp. 174–190. Springer, Heidelberg, 2006.

[39] D. Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[40] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–594, 1984.

[41] Don Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.

[42] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.

[43] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[44] Iwan M. Duursma, Pierrick Gaudry, and François Morain. Speeding up the discrete log computation on curves with automorphisms. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology – ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pp. 103–121. Springer, Heidelberg, 1999.

[45] A. E. Escott, J. C. Sager, A. P. L. Selkirk, and D. Tsapakidis. Attacking elliptic curve cryptosystems using the parallel Pollard rho method. *CryptoBytes*, 4, 1999.

[46] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pp. 329–354. Springer, Heidelberg, 1990.

[47] Fujitsu Laboratories, NICT, and Kyushu University. DL record in $\mathbb{F}_{3^{6\cdot 97}}$ of 923 bits (278 dd). NICT press release, June 18, 2012. `http://www.nict.go.jp/en/press/2012/06/18en-1.html`.

[48] Steven Galbraith. Quasi-polynomial-time algorithm for discrete logarithm in finite fields of small/medium characteristic. The Elliptic Curve Cryptography blog, June 2013. `https://ellipticnews.wordpress.com/2013/06/21`.

[49] Steven D. Galbraith. Supersingular curves in cryptography. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pp. 495–513. Springer, Heidelberg, 2001.

[50] Steven D. Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. Cryptology ePrint Archive, Report 2015/1022, 2015. `http://eprint.iacr.org/2015/1022`.

[51] Steven D. Galbraith and Raminder S. Ruprai. An improvement to the Gaudry-Schost algorithm for multidimensional discrete logarithm problems. In M. G. Parker, editor, *Cryptography and Coding*, volume 5921 of *Lecture Notes in Computer Science*, pp. 368–382. Springer, Heidelberg, 2009.

[52] Steven D. Galbraith and Raminder S. Ruprai. Using equivalence classes to accelerate solving the discrete logarithm problem in a short interval. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pp. 368–383. Springer, Heidelberg, 2010.

[53] Steven D. Galbraith, Ping Wang, and Fangguo Zhang. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. Cryptology ePrint Archive, Report 2015/605, 2015. `http://eprint.iacr.org/2015/605`.

[54] Pierrick Gaudry and Éric Schost. A low-memory parallel version of Matsuo, Chao, and Tsujii's algorithm. In D. A. Buell, editor, *Algorithmic Number Theory (ANTS-VI)*, volume 3076 of *Lecture Notes in Computer Science*, pp. 208–222. Springer, 2004.

[55] Daniel M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[56] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking '128-bit secure' supersingular binary curves - (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pp. 126–145. Springer, Heidelberg, 2014.

[57] Robert Granger, Thorsten Kleinjung, and Jens Zumbragel. Discrete logarithms in GF(2^{9234}), 2014. Announcement available at the NMBRTHRY archives, item 004666.

[58] Takuya Hayashi, Takeshi Shimoyama, Naoyuki Shinohara, and Tsuyoshi Takagi. Breaking pairing-based cryptosystems using η_T pairing over GF(3^{97}). In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pp. 43–60. Springer, Heidelberg, 2012.

[59] Takuya Hayashi, Naoyuki Shinohara, Lihua Wang, Shin'ichiro Matsuo, Masaaki Shirase, and Tsuyoshi Takagi. Solving a 676-bit discrete logarithm problem in GF(3^{6n}). In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pp. 351–367. Springer, Heidelberg, 2010.

[60] Yvonne Hitchcock, Paul Montague, Gary Carter, and Ed Dawson. The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves. *International Journal of Information Security*, 3(2):86–98, 2004.

[61] Jinhyuck Jeong and Taechan Kim. Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. Cryptology ePrint Archive, Report 2016/526, 2016. `http://eprint.iacr.org/`.

[62] Antoine Joux. Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pp. 177–193. Springer, Heidelberg, 2013.

[63] Antoine Joux. A new index calculus algorithm with complexity $L(1/4+o(1))$ in small characteristic. In T. Lange, K. Lauter, and P. Lisonek, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pp. 355–379. Springer, Heidelberg, 2014.

[64] Antoine Joux and Reynald Lercier. The function field sieve is quite special. In C. Fieker and D. R. Kohel, editors, *Algorithmic Number Theory (ANTS-V)*, volume 2369 of *Lecture Notes in Computer Science*, pp. 431–445. Springer, 2002.

[65] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Mathematics of Computation*, 72(242):953–967, 2003.

[66] Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pp. 254–270. Springer, Heidelberg, 2006.

[67] Antoine Joux, Reynald Lercier, Nigel Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pp. 326–344. Springer, Heidelberg, 2006.

[68] Antoine Joux and Cécile Pierrot. Improving the polynomial time precomputation of frobenius representation discrete logarithm algorithms - simplified setting for small characteristic finite fields. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pp. 378–397. Springer, Heidelberg, 2014.

[69] Antoine Joux and Cécile Pierrot. The special number field sieve in $\mathbb{F}_{p^n}$ - application to pairing-friendly constructions. In Z. Cao and F. Zhang, editors, *Pairing-Based Cryptography – Pairing 2013*, volume 8365 of *Lecture Notes in Computer Science*, pp. 45–61. Springer, Heidelberg, 2014.

[70] Antoine Joux and Cécile Pierrot. Nearly sparse linear algebra. Cryptology ePrint Archive, Report 2015/930, 2015. `http://eprint.iacr.org/`.

[71] Michael Kalkbrener. An upper bound on the number of monomials in determinants of sparse matrices with symbolic entries. *Mathematica Pannonica*, 8:73–82, 1997.

[72] Minkyu Kim, Jung Hee Cheon, and Jin Hong. Subset-restricted random walks for Pollard rho method on $\mathbb{F}_{p^m}$. In S. Jarecki and G. Tsudik, editors, *Public Key Cryptography – PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pp. 54–67. Springer, Heidelberg, 2009.

[73] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Advances in Cryptology – CRYPTO 2016, Part I*, Lecture Notes in Computer Science, pp. 543–571. Springer, Heidelberg, 2016.

[74] Thorsten Kleinjung. On polynomial selection for the general number field sieve. *Mathematics of Computation*, 75(256):2037–2047, 2006.

[75] Thorsten Kleinjung. Polynomial selection. Invited talk at the CADO-NFS workshop, Nancy, France, October 2008. slides available at `http://cado.gforge.inria.fr/workshop/slides/kleinjung.pdf`.

[76] Thorsten Kleinjung. Discrete logarithms in GF(2^{1279}), October 2014. Announcement available at the NMBRTHRY archives, item 004751.

[77] Neal Koblitz and Alfred Menezes. Another look at non-standard discrete log and Diffie-Hellman problems. *Journal of Mathematical Cryptology*, 2(4):311–326, 2008.

[78] Maurice Kraitchik. *Théorie des Nombres*. Gauthier–Villars, 1922.

[79] Maurice Kraitchik. *Recherches sur la Théorie des Nombres*. Gauthier–Villars, 1924.

[80] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pp. 212–229. Springer, Heidelberg, 2001.

[81] Arjen K. Lenstra. Unbelievable security: Matching AES security using public key systems (invited talk). In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pp. 67–86. Springer, Heidelberg, 2001.

[82] Arjen K. Lenstra. Key lengths. In H. Bidgoli, editor, *Handbook of Information Security*, volume 3, pp. 617–635. John Wiley & Sons, 2006.

[83] Arjen K. Lenstra, Hendrik W. Lenstra Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[84] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[85] D. V. Matyukhin. Effective version of the number field sieve for discrete logarithms in the field GF(p^k) (in Russian). *Trudy po Discretnoi Matematike*, 9:121–151, 2006.

[86] Ueli M. Maurer and Stefan Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[87] Ueli M. Maurer and Stefan Wolf. The Diffie-Hellman protocol. *Designs, Codes and Cryptography*, 19(2/3):147–171, 2000.

[88] Kevin S. McCurley. The discrete logarithm problem. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pp. 49–74. AMS, 1990.

[89] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[90] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. Characterization of elliptic curve traces under FR-reduction. In D. Won, editor, *Information Security and Cryptology – ICISC 2000*, volume 2015 of *Lecture Notes in Computer Science*, pp. 90–108. Springer, Heidelberg, 2001.

[91] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[92] B. A. Murphy. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm*. PhD thesis, Australian National University, 1999. `http://maths-people.anu.edu.au/~brent/pd/Murphy-thesis.pdf`.

[93] Brian A. Murphy. Modelling the yield of number field sieve polynomials. In J. P. Buhler, editor, *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, Lecture Notes in Computer Science, pp. 137–150. Springer Berlin Heidelberg, 1998.

[94] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology – EUROCRYPT '84*, volume 209 of *Lecture Notes in Computer Science*, pp. 224–314. Springer, Heidelberg, 1985.

[95] Hilarie Orman and Paul Hoffman. Determining strengths for public keys used for exchanging symmetric keys. Request for Comments RFC 3766, Internet Engineering Task Force (IETF), 2004.

[96] Cécile Pierrot. The multiple number field sieve with conjugation and generalized joux-lercier methods. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pp. 156–170. Springer, Heidelberg, 2015.

[97] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[98] John M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13(4):437–447, 2000.

[99] Hans-Georg Rück. On the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 68(226):805–806, 1999.

[100] Palash Sarkar and Shashank Singh. New complexity trade-offs for the (multiple) number field sieve algorithm in non-prime fields. Cryptology ePrint Archive, Report 2015/944, 2015. `http://eprint.iacr.org/2015/944`.

[101] Palash Sarkar and Shashank Singh. A general polynomial selection method and new asymptotic complexities for the tower number field sieve algorithm. Cryptology ePrint Archive, Report 2016/485, 2016. `http://eprint.iacr.org/`.

[102] Palash Sarkar and Shashank Singh. A generalisation of the conjugation method for polynomial selection for the extended tower number field sieve algorithm. Cryptology ePrint Archive, Report 2016/537, 2016. `http://eprint.iacr.org/`.

[103] Palash Sarkar and Shashank Singh. Tower number field sieve variant of a recent polynomial selection method. Cryptology ePrint Archive, Report 2016/401, 2016. `http://eprint.iacr.org/`.

[104] Takakazu Satoh and Kiyomichi Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Math. Univ. St. Pauli*, 47(1):81–92, 1998.

[105] Jürgen Sattler and Claus-Peter Schnorr. Generating random walks in groups. *Ann. Univ. Sci. Budapest. Sect. Comput.*, 6:65–79, 1985.

[106] Oliver Schirokauer. Discrete logarithms and local units. *Philosophical Transactions of the Royal Society*, 345(1676):409–423, 1993.

[107] Daniel Shanks. Class number, a theory of factorization, and genera. In D. J. Lewis, editor, *1969 Number Theory Institute*, volume 20 of *Proceedings of Symposia in Applied Mathematics*, pp. 415–440. AMS, 1971.

[108] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[109] Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, 1999.

[110] Andreas Stein and Edlyn Teske. Optimized baby step–giant step methods. *J. Ramanujan Math. Soc.*, 20(1):27–58, 2005.

[111] Douglas R. Stinson. *Cryptography: Theory and Practice*. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 3rd edition, 2006.

[112] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm, 2015. Release 2.2.0.

[113] David C. Terr. A modification of Shanks' baby-step giant-step algorithm. *Mathematics of Computation*, 69(230):767–773, 2000.

[114] Edlyn Teske. Speeding up Pollard's rho method for computing discrete logarithms. In J. P. Buhler, editor, *Algorithmic Number Theory (ANTS-III)*, volume 1423 of *Lecture Notes in Computer Science*, pp. 541–554. Springer, 1998.

[115] Emmanuel Thomé. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pp. 107–124. Springer, Heidelberg, 2001.

[116] Emmanuel Thomé. Discrete logarithms in GF(2^{607}), February 2002. Announcement available at the NMBRTHRY archives, item 001894.

[117] Emmanuel Thomé. *Algorithmes de calcul de logarithme discret dans les corps finis*. Thèse, École polytechnique, 2003. `https://tel.archives-ouvertes.fr/tel-00007532`.

[118] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[119] A. E. Western and J. C. P. Miller. *Tables of Indices and Primitive Roots*, volume 9 of *Royal Society Mathematical Tables*. Cambridge University Press, 1968.

[120] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, IT–32(1):54–62, 1986.

[121] Pavol Zajac. *Discrete Logarithm Problem in Degree Six Finite Fields*. PhD thesis, Slovak University of Technology, 2008. `http://www.kaivt.elf.stuba.sk/kaivt/Vyskum/XTRDL`.