Computing discrete logarithms in the multiplicative group of a finite field

Johannes Buchmann

December 15, 2001

Contents

Chapter 1

Introduction

Let l be a positive integer, let p be a prime number, and let $F = \mathrm{GF}(p^l)$ be the finite field of p^l elements. By F^* we denote the multiplicative group of F. It has $p^l - 1$ elements.

By the *discrete logarithm problem in* F^* we mean the following: Given $\alpha, \gamma \in F$ such that α is an element of the cyclic group $\langle\gamma\rangle$ generated by γ. Find $x \in \mathbb{N}$ such that $\alpha = \gamma^x$. We call x a *discrete logarithm* of α to base γ. Frequently, the smallest non-negative discrete logarithm is called the discrete logarithm of α to base γ. We will use this terminology if we talk about the size of a discrete logarithm and we will denote that number by $\log_\gamma \alpha$.

In our formulation of the discrete logarithm problem, we assume that a discrete logarithm of α to base γ exists, that is, $\alpha \in \langle\gamma\rangle$. Given any two elements α and γ in F^*, this is not necessarily true. In fact, a more general version of the discrete logarithm problem in F^* is the following. Given two elements $\alpha, \gamma \in F^*$. Decide, whether $\alpha \in \langle\gamma\rangle$ If the answer is "yes", find a discrete logarithm of α to base γ. However, in cryptographic applications, it is almost always clear that $\alpha \in \langle\gamma\rangle$. The cryptanalytic challenge is to find a discrete logarithm of α to base γ.

There are two types of discrete logarithm problems in the multiplicative group F^*.

1. **Full DLP** The problem of computing discrete logarithms in the full group F^*, that is, the base element has very high order.

2. **Subgroup DLP** The problem of computing discrete logarithms in a subgroup of F^*, that is, the order of the base element is a divisor of $p^l - 1$ which si considerably smaller than $p^l - 1$.

In this document we report about the state of the art of solving the full DLP and the subgroup DLP. This report is organized as follows. In Chapter

2 we review generic algorithms which are used to attack the subgroup DLP. In Chapter 3 we review index calculus algorithms which are used to solve the full DL. In Chapter 4 we briefly discuss quantum algorithms. Finally, in Chapter 5 we draw conclusions.

In this report we have extensively used the reports of Odlyzko [17] and Denny, Weber, and Schirokauer [25]. We have also used explanations from [4].

Chapter 2

Generic algorithms

In this chapter, we discuss generic discrete logarithm algorithms that work in any cyclic group G as long as computing in that group is efficiently possible. To be more precise, we assume that the following is efficiently possible:

1. We have a representation of the group elements of binary length $O(|G|)$ where $|G|$ denotes the cardinality of G and the *binary length* of an object is the number of bits required to represent that object. Sorting those representations is possible.

2. Given two elements, we can decide whether they are equal and we can multiply and divide the elements.

In F^*, elements are uniquely represented as coefficient vectors with respect to a GF(p)-basis of F. The entries in that vector are in GF(p). Equality testing means comparing the representing vectors. Multiplication and division uses multiplication tables for the basis elements. Those operations can be efficiently implemented.

Let $\alpha, \gamma \in F^*$ such that $\alpha \in \langle\gamma\rangle$. Let $G = \langle\gamma\rangle$ and let n be the order of G. If n is not known, then n is replaced by an upper bound on the cardinality of G, for example, $n = p^l$. Recall that it is our goal to find $x \in \mathbb{N}$ such that

$$\alpha = \gamma^x. \tag{2.1}$$

2.1 Enumeration

The simplest method for solving the DL problem is to test whether $x = 0, 1, 2, 3, \ldots$ satisfy (2.1). As soon as the answer is "yes", a discrete logarithm is found. This is called *enumeration*. Enumeration requires $x-1$ multiplications and x comparisons in G. Only the elements α, γ and γ^x need to be stored. Hence, enumeration

only requires space for three group elements. In practice, applying enumeration is infeasible if the finite field F has more that 2^{80} elements.

2.2 Shanks "baby-step giant-step" technique

A considerable improvement of the enumeration algorithm is the *baby-step giant-step algorithm* of D. Shanks (see [27]). This algorithm requires fewer group operations but more storage. We describe this algorithm.

Set

$$u = \lceil \sqrt{n} \rceil$$

and write the unknown discrete logarithm x as

$$x = qu + r, \quad 0 \leq r < m.$$

Then

$$(\gamma^u)^q = \alpha\gamma^{-r}.$$

The baby-step giant-step algorithms computes the set of *baby-steps*

$$B = \{(\alpha\gamma^{-r}, r) : 0 \leq r < u\}.$$

If this set contains a pair $(1, r)$, then $\alpha\gamma^{-r} = 1$ (i.e., $\alpha = \gamma^r$). Hence, $x = r$ is a discrete logarithm of α to base γ. If B contains no such pair, the algorithm determines

$$\delta = \gamma^u.$$

Then it tests for $q = 1, 2, 3, \ldots$ whether the group element δ^q is the first component of an element in B (i.e., whether there is a pair (δ^q, r) in B). As soon as this is true, we have

$$\alpha\gamma^{-r} = \delta^q = \gamma^{qu}$$

which implies

$$\alpha = \gamma^{qu+r}.$$

Therefore, a discrete logarithm is

$$x = qu + r.$$

The elements δ^q, $q = 1, 2, 3 \ldots$ are called *giant-steps*. The algorithm compares each δ^q with all first components of the baby-step set B. To make this comparison efficient, the elements of B are stored in a hash table where the key is the first entry in an element of B (see [6], Chapter 12).

This algorithm has running time $O(\sqrt{n})$ and requires storage $O(\sqrt{n})$. If the discrete logarithm is known to lie in an interval of length at most m, then a variant of Shanks' algorithm has running time $O(m^{1/2})$ and needs storage $O(m^{1/2})$ (see [5]).

The most serious problem with Shanks' algorithm is the storage requirement. It makes the Shanks algorithm inapplicable for $n > 2^{120}$.

2.3 Variants of the Shanks "baby-step giant-step" technique

The algorithms of Buchmann, Jacobson, and Teske [5] and of Terr [30] are variants of the Shanks babystep-giantstep technique whose running time and storage requirement depends on the actual value x of the discrete logarithm rather than the order of the cyclic group generated by γ. Like enumeration, those algorithms are fast if the discrete logarithm is small. They have running time and storage requirement $O(\sqrt{x})$. In fact, the algorithm of Buchmann, Jacobson, and Teske requires at most $4\sqrt{x}$ group operations and space for at most $2\sqrt{x}$ group elements. This algorithm guesses an upper bound for the discrete logarithm. If the discrete logarithm is not found, then the upper bound is doubled. Terr's algorithm calculates one baby step followed by one giant-step and is approximately as efficient as the algorithm of Buchmann, Jacobson and Teske. Note that the O-constants are very small.

Those algorithms show that in order for the DL problem to be intractable, it does not suffice that the the cyclic group generated by the base element is large, but the discrete logarithm itself must be large.

The modified algorithms from this Section are inapplicable for $x < 2^{120}$.

2.4 The Pollard algorithms

There are several algorithms due to John Pollard which avoid the space problems of the Shanks algorithm and its variants by introducing randomness. As an example, we sketch the Pollard rho algorithm. That algorithm simulates a random walk in the group and uses the birthday paradox to determine the discrete logarithm after approximately $\sqrt{n}$ steps.

Again, we want to solve the DL problem (2.1). We need three pairwise disjoint subsets G_1, G_2, G_3 of G such that $G_1 \cup G_2 \cup G_3 = G$. Let $f : G \to G$ be defined by

$$f(\beta) = \begin{cases} \gamma\beta & \text{if } \beta \in G_1, \\ \beta^2 & \text{if } \beta \in G_2, \\ \alpha\beta & \text{if } \beta \in G_3. \end{cases}$$

We choose a random number x_0 in the set $\{1, \ldots, n\}$ and compute the group element $\beta_0 = \gamma^{x_0}$. Then, we compute the sequence (β_i) by the recursion

$$\beta_{i+1} = f(\beta_i).$$

The elements of this sequence can be written as

$$\beta_i = \gamma^{x_i}\alpha^{y_i}, \quad i \geq 0.$$

Here, x_0 is the initial random number, $y_0 = 0$, and we have

$$x_{i+1} = \begin{cases} x_i + 1 \mod n & \text{if } \beta_i \in G_1, \\ 2x_i \mod n & \text{if } \beta_i \in G_2, \\ x_i & \text{if } \beta_i \in G_3, \end{cases}$$

and

$$y_{i+1} = \begin{cases} y_i & \text{if } \beta_i \in G_1, \\ 2y_i \mod n & \text{if } \beta_i \in G_2, \\ y_i + 1 \mod n & \text{if } \beta_i \in G_3. \end{cases}$$

Since we are working in a finite group, two elements in the sequence (β_i) must be equal (i.e., there is $i \geq 0$ and $k \geq 1$ with $\beta_{i+k} = \beta_i$). This implies

$$\gamma^{x_i} \alpha^{y_i} = \gamma^{x_{i+k}} \alpha^{y_{i+k}}$$

and therefore

$$\gamma^{x_i - x_{i+k}} = \alpha^{y_{i+k} - y_i}.$$

Hence, the discrete logarithm x of α to the base γ satisfies

$$(x_i - x_{i+k}) \equiv x(y_{i+k} - y_i) \mod n.$$

We solve this congruence. The solution is unique mod n if $y_{i+k} - y_i$ is invertible mod n. If the solution is not unique, then the discrete logarithm can be found by testing the different possibilities mod n. If there are too many possibilities, then the algorithm is applied again with a different initial x_0.

We estimate the number of elements β_i that must be computed before a *match* is found (i.e., a pair $(i, i+k)$ of indices for which $\beta_{i+k} = \beta_i$). For this purpose, we use the birthday paradox. The possible birthdays are the group elements. We assume that the elements of the sequence $(\beta_i)_{i \geq 0}$ are random group elements. This is obviously not true, but the construction of the sequence makes it very similar to a random sequence. Therefore, $O(\sqrt{|G|})$ sequence elements are sufficient to make the probability for a match greater than $1/2$. For details about the birthday paradox see [4] 4.3.

Thus far, our algorithm must store all triplets (β_i, x_i, y_i). As we have seen, the number of elements of the sequence is of the order of magnitude $\sqrt{|G|}$, as in Shanks' baby-step giant-step algorithm. But we will now show that it suffices to store a single triplet. Therefore, the Pollard rho algorithm is much more space efficient than the baby-step giant-step algorithm.

Initially, (β_1, x_1, y_1) is stored. Now suppose that at a certain point in the algorithm, (β_i, x_i, y_i) is stored. Then (β_j, x_j, y_j) is computed for $j = i+1, i+2, \ldots$ until either a match is found or $j = 2i$. In the latter case, we delete β_i and store β_{2i}. Hence, we only store the triplets (β_i, x_i, y_i) with $i = 2^k$.

We explain how a match is found if only one triplet is stored. Denote by i the index of the triplet that is currently stored. If $i = 2^j \geq s$, then β_i is in the period. In addition, if $2^j \geq k$, then the sequence

$$\beta_{2^j+1}, \beta_{2^j+2}, \ldots, \beta_{2^{j+1}}$$

is at least as long as the period. One of its elements is equal to β_{2^j}. But this is exactly the sequence that is computed after b_{2^j} has been stored. All of its elements are compared with β_{2^j}. Hence, one of these comparisons will reveal a match. Because the sum of the lengths of the pre-period and the period is $O(\sqrt{|G|})$, it follows that the number of sequence elements that must be computed before a match is found is $O(\sqrt{|G|})$. Therefore, the algorithm has running time $O(\sqrt{|G|})$ and needs space for $O(1)$ triplets. This is much more space efficient than the baby-step giant-step algorithm.

The algorithm is even more efficient if eight triplets are stored. This works as follows. Initially, all eight triplets are equal to (β_0, x_0, y_0). Then those triplets are successively replaced. Let i be the index of the last stored triplet. Initially, we have $i = 1$. For $j = 1, 2, \ldots$ we compute (β_j, x_j, y_j) and do the following:

1. If β_j is equal to one of the stored group elements, then a match is found and the computation of the sequence terminates.
2. If $j \geq 3i$, then the first of the eight triplets is deleted and (β_j, x_j, y_j) is the new last triplet.

Further improvements are due to Teske [31]. The Kangaroo method of Pollard [21] is another variant of the rho method. Its running time is $O(m^{1/2})$ when the discrete logarithm is known to lie in an interval of length at most m. The space requirement is $O(1)$.

Algorithm	running time	storage
Enumeration	n	$O(n)$
Shanks [27]	$O(\sqrt{n})$	$O(\sqrt{n})$
BJT [5]	$6\sqrt{m}$	$2\sqrt{m}$
Terr [30]	$O(\sqrt{x})$	$O(\sqrt{x})$
Pollard rho [22]	$O(\sqrt{n}$	$O(1)$
Pollard kangaroo [21]	$O(\sqrt{m}$	$O(1)$

Table 2.1: Generic algorithms, DL x in group of order n, known to lie in interval of length m

2.5 Pohlig-Hellman algorithm

If the factorization of the order $n = p^l - 1$ of $\mathrm{GF}(p^l)^*$ is known, then the Pohlig-Hellman algorithm [19] reduces the discrete logarithm problem in $\mathrm{GF}(p^l)^*$ in polynomial time to discrete logarithm problems in subgroups of prime order q where q are the prime divisors of n. First, finding a discrete logarithm x is reduced to the determination of a discrete logarithm x_q modulo the prime powers q^{e_q} which divide n. Next, each x_q is written in a q-adic expansion and

each digit in this expansion is calculated as a discrete logarithm in some group of order q. In those groups, one of the algorithms from the previous sections can be applied. The running time of the Pohlig-Hellman algorithm combined with the Pollard rho algorithm is approximately $\sqrt{q}$ where q is the largest prime divisor of n. The storage requirement is negligible.

2.6 Parallelization

There are parallel versions of the Pollard kangaroo method [21], [32]. The elapsed time for the computation shrinks by a factor linear in the number of processors used. However, substantial storage space has to be available at a central location (see [14]).

2.7 Experimental data

Since in general, index calculus methods compute discrete logarithms in finite fields much faster than the known generic methods, no extensive experimental results are available. However, in the group of points of an elliptic curve over a finite field, only generic methods are available to compute discrete logarithms. Using the parallel Pollard kangaroo method, it was possible to compute discrete logarithms in an elliptic curve point group over a finite prime field where the group order is a 97-bit prime using $2 * 10^{14}$ group operations (see [8]). Also, it was possible to compute the discrete logarithm in an elliptic curve point group with a 108-bit order over a finite field of characteristic 2 using $2.3 * 10^{15}$ group operations (see [8]).

2.8 Lower bounds

There are lower bounds on the complexity of the discrete logarithm problem. In a group where the group operations including equality testing has to be performed by oracle queries, discrete logarithm computations require at least p operations where p is the largest prime divisor of the group order (see [2]). If group elements have a unique encoding (which allows sorting in the Shanks algorithm and randomizing in Pollard's algorithms) but group operations have still to be carried out using oracle queries, then Nechaev [16] and Shoup [29] prove a lower bound of order $p^{1/2}$.

2.9 Conclusions

In this section, we discuss the following question: How do we have to choose γ and α in (2.1) such that solving the discrete logarithm problem using a generic

algorithm is infeasible. This is relevant for the security of cryptographic schemes such as the Schnorr identification and signature scheme [26] and the DSA [9], [36].

Let the order of the base element in (2.1) be n. To protect against Pohlig-Hellman attacks, n is a prime number. Also, the discrete logarithm x of α to base γ is of order n. Then the known generic algorithms execute at least $n^{1/2}$ group operations to find the discrete logarithm x. Lenstra and Verheul [14], assuming technological and algorithmic progress, predict secure group orders as a function of time. This is shown in Table 2.2.

Year	Group order
2002	127
2010	138
2020	151
2030	165
2040	179
2050	193

Table 2.2: Secure cryptographic group orders

This means that the discrete logarithm problem in a cyclic subgroup of prime order $q \sim 2^{160}$, as used in the DSS [9], is intractable until 2020. Although the lower bounds quoted in Section 2.8 indicate that that this is a pretty safe assertion, mathematical progress that uses the special structure of such subgroups may very well make this discrete logarithm problem much easier.

Chapter 3

Index calculus algorithms

In this chapter, we describe index calculus algorithms. Those algorithms use the special structure of the finite field F. The basic idea of hat algorithm goes back to Kraitchik (see [15]). Currently, the most efficient variants of the index calculus method are the Number Field Sieve (NFS) and the Function Field Sieve (FFS) on which we concentrate in this chapter.

Let p be a prime number, $l \in \mathbb{N}$ and $q = p^l$. Also, let $\gamma \in \mathrm{GF}(q)^*$.

3.1 Idea

The basic idea of index calculus algorithms is that if

$$\prod_{i=1}^{m} \alpha_i^{e_i} = \gamma^r \tag{3.1}$$

for some elements α_i in the subgroup generated by γ and exponents $e_i, r \in \mathbb{Z}$, $1 \leq i \leq m$, then

$$\sum_{i=1}^{m} e_i \log_\gamma \alpha_i \equiv r \bmod q - 1. \tag{3.2}$$

Many *relations* of the type (3.1) yield a system of linear congruences of type (3.2). This system (3.2) yields the discrete logarithms of the α_i if it is non-singular.

There are two computational problems in the index calculus algorithm:

1. Find sufficiently many relations.
2. Solve the linear system.

We explain the computation of relations in a special case. Suppose that $l = 1$ so $q = p$ and $\mathrm{GF}(q) = \mathrm{GF}(p) = \mathbb{Z}/p\mathbb{Z}$ is a prime field. Let $\gamma = c + p\mathbb{Z}$ and $\alpha = a + p\mathbb{Z}$, that is, c is a representative of the residue class γ and a is a representative of the residue class α. A simple approach for finding relations is to take a random integer r, compute $u = g^r \bmod p$, and check whether

$$u = \prod_{i=1}^{m} p_i^{e_i}, \tag{3.3}$$

where the p_i are prime numbers satisfying $p_i < B$ for some bound B. When (3.3) is satisfied then u is called *B-smooth*. From (3.3) we obtain

$$\prod_{i=1}^{m} p_i^{e_i} \equiv g^r \bmod p \tag{3.4}$$

which is a relation.

In the general case, the known index calculus algorithms work as follows (see [25]).

Let R be a Dedekind domain. In R, every ideal factors uniquely into prime ideals. Assume that there is a surjective ring homomorphism

$$\phi : R \to \mathrm{GF}(q).$$

Let S be a set of prime ideals of R and call an element of R smooth if the ideal it generates factors over S. The set S is called the *factor base*. Let $c \in R$ such that $\phi(c) = \gamma$. For an element $r \in R$ we denote by (r) the ideal generated by r. The algorithms find many pairs $(u, v) \in R^2$ such that

1. $\phi(u) = \phi(v)$.
2. (u) factors into a power of (c) and a smooth element, that is, $(u) = (c)^x \prod_{P \in S} P^{e(P)}$.
3. v is smooth, that is, $(v) = \prod_{P \in S} P^{f(P)}$.

Those pairs and linear algebra techniques are used to solve the discrete logarithm problem. It is also possible to use two different Dedekind domains R_i, $i = 1, 2$ (see [13]).

3.2 Asymptotic running times

3.2.1 The function $L_x[u, v]$

To estimate the running time and storage requirement of the index calculus algorithms the function

$$L_x[u, v] = e^{v(\log x)^u (\log\log x)^{1-u}}$$

is used, where x, u, v are positive real numbers. I explain the meaning of this function. We have

$$L_x[0, v] = e^{v(\log x)^0 (\log\log x)^1} = (\log x)^v \tag{3.5}$$

and

$$L_x[1, v] = e^{v(\log x^1 (\log\log x)^0} = e^{v \log x}. \tag{3.6}$$

Let x be a positive integer which is the input for an algorithm. In the context of this report, x is the cardinality of the finite field GF(q).

If an algorithm has running time $L_x[0, v]$, then by (3.5) it is a polynomial time algorithm. Its complexity is bounded by a polynomial in the size of the input. The algorithm is considered efficient, although its real efficiency depends on the degree v of the polynomial.

If the algorithm has running time $L_x[1, v]$, then by (3.6) it is exponential. Its complexity is bounded by an exponential function in the length of the input. The algorithm is considered inefficient.

If the algorithm has running time $L_x[u, v]$ with $0 < u < 1$, then it is *subexponential.* The algorithm is slower than polynomial but faster than exponential. So the function $L_x[u, v]$ can be viewed as a linear interpolation between polynomial time and exponential time.

3.2.2 Rigorous algorithms

We describe the knowledge concerning discrete logarithm algorithms whose running times can be proved.

In the case that $q = p^l$ with $p \leq l^{o(l)}$, Pomerance et al. [3] prove that the index calculus algorithm that uses as the ring R a polynomial ring (IC-PR algorithm) has running time $L_q[1/2, \sqrt{2} + o(1)]$.

In the case that $q = p$, Pomerance [23] proves for $R = \mathbb{Z}$ a running time $L_q[1/2, \sqrt{2} + o(1)]$.

In the case that $q = p^2$ Lovorn-Bender [3] proves a running time $L_q[1/2, 3/2 + o(1)]$. Here R is a maximal quadratic order.

3.2.3 Heuristic algorithms

There are many discrete logarithm algorithms that have not been rigorously analyzed yet. However, those algorithms can be analyzed heuristically. In a heuristic analysis, unproved but plausible assumptions are used such as the assumption of some generalized Riemann hypothesis. All algorithms in this section are of this kind.

In the case that $p > l$, Adleman and DeMarrais prove a running time $L_q[1/2, 2 + o(1)]$. Here R is a number ring (IC-NR algorithm). This result

together with the analysis of the IC-PR algorithm shows that discrete logarithms in any GF(q) can be computed in time $L_q[1/2, c+o(1)]$ where $c \leq 2$ and $q \to \infty$.

Asymptotically much faster are the Number Field Sieve (NFS) and the Function Field Sieve (FFS).

In the case that $l < (\log p)^{1/2} + \varepsilon$ for any $\varepsilon > 0$, the NFS has a conjectured running time of $L_q[1/3, c+o(1)]$ with $c = (64/9)^{1/3} = 1.9229...$ (see [24]).

For special primes p, for example for $p = r^n + a$, where r and a are small and n is large, versions of the number field sieve for the fields GF(p) we have $C = 1.5262...$ or even less. Those algorithms are called Special Number Field Sieve (SNFS) (see [24]).

In the case that $l \geq (\log p)^2$ the FFS has a conjectured running time of $L_q[1/3, (32/9)^{1/3} + o(1)]$ (see [1]).

A gap exists between the ranges of fields for which the NFS and the FFS yield a $L_q[1/3, c+o(1)]$-algorithm and the problem of finding an algorithm with such a conjectured running time for all fields remains open.

3.3 Implementation

In this section we discuss various issues concerning the implementation of the NFS. The NFS and the FFS are currently the most efficient DL algorithm in finite fields. Extensive experiments have been made in prime fields GF(p) and in fields of characteristic 2 GF(2^l).

3.3.1 Polynomial

If the NFS is used in a prime field GF(p) then two irreducible polynomials g, g' in $\mathbb{Z}[X]$ have to be found that have a common root mod p. For example, in the record computation described in Section 3.3.4 one polynomial is of degree 2 and the other is of degree 3. To speed up the NFS, it is necessary to make the coefficients of the polynomials as small as possible. In [13] LLL-reduction is used to find good polynomials. The coefficient size is $p^{1/(d+1)}$. The NFS algorithm is very sensitive to the size of those coefficients. If the prime p is of a special form, then finding a good polynomial may be much easier. For example, for the prime $p = (739 \cdot 7^{149} - 736)/3$ the polynomial $f(X) = 739X^5 - 5152$ can be used (see [33]).

If the FFS is used to find discrete logarithms in GF(2^l), the situation is analogous. The rational integers are replaced by the polynomials over GF(2). The prime p is replaced by a irreducible polynomial over GF(2) of degree l.

3.3.2 Sieving

In the sieving stage, elements of smooth norm must be found in the number fields or function fields defined by g and g'. Sieving algorithms have the advantage that they require no trial division. Once it is known that an element or ideal under consideration is divisible by a prime number or a prime ideal, it can be deduced which are the other elements or ideals that are divisible by that number or ideal. The most efficient method for this sieving stage is the lattice sieve (see [20] and [34]).

3.3.3 Linear algebra

The linear systems that arise in the context of the index calculus algorithms are extremely sparse since in the relations only very few exponents are non-zero. Still, the systems are extremely large. For example, in the record computation [12], a linear system with 2685597 equations and 1242551 unknowns had to be solved.

The linear algebra part has two steps. The first step is structured Gaussian elimination (see [17]. It reduces the size of the system while keeping its sparseness. For example, equations in which only one variable has a non-zero coefficient can be eliminated. They determine that variable. So the corresponding variable can also be eliminated.

The second step uses Lanczos's algorithm (see [10]). It is a solver for linear systems which makes use of the sparseness of the system. It heavily uses matrix times vector computations which are cheap when the matrix is sparse. That algorithm can be parallelized (see [7]). A serious problem of the linear algebra step is the size of the entries of the corresponding matrix. Those entries are elements of the finite field in which the DL problem is solved. This size makes the linear algebra problem much more difficult than the corresponding problem in integer factoring algorithms where the entries are either 0 or 1. However, modern computers allow solving fairly large systems.

3.3.4 Record in prime fields

The fastest algorithm for computing discrete logarithms in GF(p) is the Number Field Sieve (see [24] and [13]).

The current record is the solution of the McCurley challenge problem [15] to compute the discrete logarithm modulo a prime of 129 decimal digits by Denny and Weber [35]. However, the prime involved was of a special form so that the Special Number Field sieve could be applied.

The current record for the general number field sieve is the computation of discrete logarithms modulo a 120 digits prime. This was done in 10 weeks, on a unique 525MHz quadri-processors Digital Alpha Server 8400 computer by Joux

and Lercier [12].

After a 40 days computation on a quadri-processor Alpha Server 8400 computer, the necessary relations were found. The corresponding linear system has 2685597 equations with 1242551 unknowns.

The linear algebra step was carried out as follows. First, structured Gaussian eliminations reduced the system to 271654 equations in 271552 unknowns with 22690782 non null entries. The time needed for this on only one processor was less than 1 day.

Then, the critical phase was the final computation via Lanczos's algorithm [10]. The parallelized version of this algorithm took 30 days on 4 processors.

3.3.5 Record in fields of characteristic 2

The most efficient algorithm for computing discrete logarithms in fields of characteristic 2 is the Function Field Sieve [1].

The current record is the computation of discrete logarithms in $GF(2^{521})$. This was done in one month on a unique 525MHz quadri-processors Digital Alpha Server 8400 computer by Joux and Lercier [11].

After a 3 weeks computation on a quadri-processor Alpha Server 8400 computer, the necessary relations were found. The corresponding system has 472121 equations with 450940 unknowns.

The linear algebra step was carried out as follows. First, structured Gaussian eliminations reduced the system to 197039 equations in 196939 unknowns with 12220108 non null entries. The time needed for this on only one processor was less than 1 day.

Then, the critical phase was the final computation via Lanczos's algorithm [10]. The parallelized version of this algorithm took 10 days on 4 processors.

3.4 Conclusions

Taking the algorithms presented in this chapter and certain algorithmic progress into account, Lenstra and Verheul suggest field sizes for secure crypto systems as a function of time. This is shown in Table 3.1.

However, it may always be possible that some mathematician invents a completely new and much more efficient technique for finding relations on much smaller factor bases. The invention of the Number Field Sieve by Pollard is an example for such a mathematical break through which was not anticipated. In that case, the predictions of Lenstra and Verheul may turn out to be far too optimistic.

Year	Field size
2002	1028
2010	1369
2020	1881
2030	2493
2040	3214
2050	4047

Table 3.1: Secure cryptographic field sizes

Chapter 4

Quantum algorithms

A dangerous long-term threat to discrete logarithm cryptosystems comes from quantum computers. Shor [28] showed that if such machines could be built, discrete logs in finite fields could be computed in polynomial time. Recent results on quantum computing can be found in the archive [18]. While there is still some debate on whether quantum computers are feasible, no fundamental obstructions to their construction have been found, and novel approaches are regularly suggested. However, all experts agree that even if quantum computers are eventually built, it will take many years to do so (see [17]).

Chapter 5

Conclusions

As of today, the full discrete logarithm is best attacked by index calculus algorithms and the subgroup discrete logarithm problem is best solved by generic algorithms. Using the predictions of [14] it seems to be appropriate to use the sizes in Table 5.1 in order for the discrete logarithm problem in finite fields to be intractable.

Year	Field size	Subgroup size
2002	1028	127
2010	1369	138
2020	1881	151
2030	2493	165
2040	3214	179
2050	4047	193

Table 5.1: Sizes that make the DL in finite fields intractable.

Those predictions take moderate algorithmic progress into account. However, there are two serious threats that might make them totally invalid:

1. Quantum computers. If they can be built, then discrete logarithms in finite fields can be computed in polynomial time.
2. Mathematical breakthroughs. There is no proof that the DL problem in prime fields remains difficult. There is always the possibility that mathematicians find much faster DL algorithms. Even polynomial time DL algorithms are not impossible.

We apply those conclusions to the security analysis of the Digital Signature Algorithm (DSA) [9], [36]. The DSA is only secure, if the full discrete logarithm problem is intractable in a prime field that has between 2^{512} and 2^{1024} elements

and if the subgroup discrete logarithm problem is intractable in a subgroup with approximately 2^{160} elements. In view of Table 5.1 the DSA with those parameters will soon become insecure. To maintain security for the next 20 years, it is recommended to increase the field size to 2^{1900}. This does not take unexpected break throughs into account. I recommend using the DSA with larger primes in an environment where it can easily be replaced by another signature scheme, if necessary.

Bibliography

[1] Adleman, L. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *ANTS I* (1994), L. Adleman and M.-D. Huang, Eds., vol. 877 of *Lecture Notes in Computer Science*.

[2] Babai, L., and Szemeredi, E. On the complexity of matrix group problems. In *Proc. 25th FOCS* (1984), IEEE Press, pp. 229–240.

[3] Bender, R. L., and Pomerance, C. Rigourous discrete logarithm compitations in finite fields via smooth polynomials. In *Computational Perspectives on Number Theory (Chicago 1995)* (1998), vol. 7 of *AMS/IS Stud. Adv. Math, Amer. Math. Soc.*, pp. 221–232.

[4] Buchmann, J. *Introduction to Cryptography*. Springer-Verlag, New York, 2001.

[5] Buchmann, J., Jacobson, Jr., M., and Teske, E. On some computational problems in finite abelian groups. *Mathematics of Computation 66* (1997), 1663–1687.

[6] Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms*. MIT Press, Cambridge, Massachudetts, 1990.

[7] Denny, T. *Lösen dünn besetzter Gleichungssysteme über endlichen Körpern*. PhD thesis, Universität des Saarlandes, 1997.

[8] Escott, A. E., Sager, J. C., and Selkirk, A. P. L. Attacking elliptic curve cryptosystems using the parallel pollard rho method. *Cryptobytes 4* (1999), 15–19.

[9] FIPS 186-2, Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 2000.

[10] Golub, G., and van Loan, C. *Matrix computations*. The John Hopkins University Press, 1989.

[11] JOUX, A., AND LERCIER, R. Discrete logarithms in GF(2^n). Announcement on the NMBRTHRY Mailing List, 25. September 2001.

[12] JOUX, A., AND LERCIER, R. Discrete logarithms in GF(p). Announcement on the NMBRTHRY Mailing List, 17. April 2001.

[13] JOUX, A., AND LERCIER, R. Improvements to the general number field sieve for discrete logarithms in prime fields. *Math. Comp.* (2001). To appear.

[14] LENSTRA, A., AND E.R.VERHEUL. Selecting cryptographic key sizes, October 1999.

[15] MCCURLEY, K. The discrete logarithm problem. In *Cryptography and computational number theory, Proc. Symp. Apll. Math* (1990), C. Pomerance, Ed., vol. 42 of *Amer. Math. Soc.*, pp. 49–74.

[16] NECHAEV, V. On the complexity of a deterministic algorithm for a discrete logarithm. *Math. Notes 55* (1994), 165–172.

[17] ODLYZKO, A. Discrete logarithms: The past and the future. *Design, Codes and Cryptography 19* (2000), 129–15.

[18] PHYSICS E-PRINT ARCHIVE, Q. `http://xxx.lanl.gov/archive/quant-ph`.

[19] POHLIG, S. C., AND HELLMAN, M. E. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. 106–110.

[20] POLLARD, J. The lattice sieve. In *The development of the number field sieve*, A. K. Lenstra and H. W. Lenstra, Jr., Eds., no. 1554 in Lecture Notes in Mathematics. Springer-Verlag, 1993, pp. 43–49.

[21] POLLARD, J. Kagaroos, monopoly and discrete logarithms. *J. Cryptology 13* (2000), 437–447.

[22] POLLARD, J. M. Monte Carlo methods for index computation (mod p). 918–924.

[23] POMERANCE, C. Fast rigorous factorization and discrete logarithm algorithms. In *Discrete algorithms and complexity*, D. Johnson, T. Nishizeki, A. Nozaki, and H. Wilf, Eds. Academic Press, 1987, pp. 119 – 143.

[24] SCHIROKAUER, O. Discrete logarithms and local units. *Phil. Trans. R. Soc. London A 345* (1993), 409–423.

[25] SCHIROKAUER, O., WEBER, D., AND DENNY, T. Discrete logarithms: the effectiveness of the index calculus method. In *ANTS II* (Berlin, 1996), H. Cohen, Ed., vol. 1122 of *Lecture Notes in Computer Science*, Springer-Verlag.

[26] SCHNORR, C. Efficient signature generation by smart cards. In *Advances in Cryptology - CRYPTO 89* (1991), Lecture Notes in Computer Science, Springer - Verlag, pp. 161–174.

[27] SHANKS, D. Class number, a theory of factorization and genera. In *Proc. Symp. Pure Math. 20* (1971), AMS, Providence, R.I., pp. 415–440.

[28] SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Computing 26* (1997), 1484–1509.

[29] SHOUP, V. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology - Eurocrypt 97* (1997), W. Fumy, Ed., vol. 1233 of *LNCS*, Springer-Verlag, pp. 256–266.

[30] TERR, D. C. A modification of Shanks' baby-step giant-step algorithm. *Math. Comp. 69*, 230 (2000), 767–773.

[31] TESKE, E. Speeding up pollard's rho method for computing discrete logarithms. In *Algorithmic Number Theory: Third Intern. Symp., ANTS-III* (1999), J.P.Buhler, Ed., vol. 1423 of *LNCS*, Springer-Verlag, pp. 541–554.

[32] VAN OORSCHOT, P., AND WIENER, M. Parallel collision search with cryptanalytic applications. *J. Cryptology 12* (1999), 1–28.

[33] WEBER, D. *On the computation of discrete logarithms in finite prime fields.* PhD thesis, Universität des Saarlandes, 1997.

[34] WEBER, D. *On the computation of discrete logarithms in finite prime fields.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[35] WEBER, D., AND DENNY, T. The solution of mccurleys discrete logarithm challenge. In *Advances in Cryptology - Crypto 98* (1998), vol. 1462 of *LNCS*, pp. 458–471.

[36] ANSI X9.30:1-1997, Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA). Available from the ANSI X9 Catalog, 1997.

Index