Evaluation Report on the **Discrete Logarithm** Problem over finite fields

Jacques Stern

1 Introduction

This document is an evaluation of the discrete logarithm problem over finite fields (DLP), as a basis for designing cryptographic schemes. It relies on the analysis of numerous research papers on the subject.

The present report is organized as follows: firstly, we review the DLP and several related problems such as the Diffie-Hellman problem. Next, we analyze the various algorithms that are currently known to solve the problem. For each algorithm, we study its asymptotic behaviour, as well as its practical running time, based on experiments reported in the literature. Finally, we derive consequences in terms of key sizes for cryptosystems whose security depend on the hardness of the DLP. We conclude by making some predictions on how the key sizes might evolve. This is as requested by IPA.

2 The DLP and related problems

In this section, we review the DLP and several related problems, such as the computational and decisional Diffie-Hellman problems and we investigate their security in terms of complexity-theoretic reductions.

2.1 The DLP as a cryptographic primitive

The discrete logarithm problem in a finite group G can be stated as follows: compute x from g and $u = g^x$. Integer x is called the discrete logarithm of u in base g, $x = \log_g(u)$. Of particular interest in the present report, is the case when G is the multiplicative group of a a finite field $\mathbb{K}$ with $q = p^n$ elements, where p is a prime. In this setting, cryptographic applications are almost exclusively related with the case $p = 2$ and the case $n = 1$. The first case corresponds to a field of characteristic 2, while the second uses the group $\mathbb{Z}_p^\star$ of invertible elements of $\mathbb{Z}_p$. Each case is an instance of a larger family:

fields of small characteristic p and fields of small degree n, respectively. Subgroups are also used. For example, p can be chosen such that $p-1$ has a large prime factor q of prescribed size. In this case, there is a subgroup of order q, consisting of elements x in $\mathbb{Z}_p^\star$ such that $x^{\frac{p-1}{q}} = 1 \bmod p$.

The basic security assumption on which cryptosystems based on the DLP rely is *one-wayness* (OW): an attacker cannot recover x from g and g^x. In a more precise complexity-theoretic framework, this means that the success probability $\mathsf{Succ}^{\mathsf{dlp}}(\mathcal{A})$ of any polynomial time adversary $\mathcal{A}$ attempting to invert $x :\longrightarrow g^x$ is negligible, i.e. asymptotically smaller than the inverse of any polynomial function of the security parameter k. The security parameter can be taken as the degree n of the field, for small characteristic, and as the bit-size of p for small degree. Probabilities are taken over g and x. In the case of fields of small degree, it is implicit that a prime generation algorithm $\mathcal{K}(1^k)$ produces p. It uses randoin coins Ω and probabilities also range over these random coins.

To go from asymptotic to exact estimates, we can define $\mathsf{Succ}^{\mathsf{dlp}}(\tau, k)$ as the probability for an adversary to find the preimage of a given element within time τ. We turn the above into symbols, in the case of prime fields ($n = 1$):

$$\mathsf{Succ}^{\mathsf{dlp}}(\mathcal{A}) = \Pr[p \leftarrow \mathcal{K}(1^k), g \in_R G, x \in_R \{0, \ldots, p-1\}, u \leftarrow (g^x \bmod p) : \mathcal{A}(p, u) = x].$$

The hardness of the DLP is the statement that, for large enough k, this probability is extremely small.

Many cryptographic schemes rely on the assumption that the DLP is hard. To name a few, the El Gamal public key cryptosystem and signature scheme [10], the Digital Signature Algorithm DSA [9], the Cramer-Shoup cryptosystem [8, 35].

2.2 The Diffie-Hellman problem

The modern approach to cryptographic design relies on the notion of security proof. Such proofs are *reductions* in the sense of complexity theory. Given an attacker $\mathcal{A}$ that breaks the cryptographic scheme, one designs another machine $\mathcal{B}$, solving the underlying hard problem. The relation between running times and success probabilities of $\mathcal{A}$ and $\mathcal{B}$ is further made explicit, so that, if the security loss is not too large, concrete estimates on key sizes may be derived from the proof.

Ideally, one would like to establish the security of a scheme based on the sole assumption that the underlying problem is hard. Unfortunately, very few schemes allow such a proof. One of them is the Cramer-Shoup cryptosystem, quoted above [8]. For the others, the best one can hope for, is a proof carried in a non-standard computational model, as proposed by Bellare and Rogaway [2], following an earlier suggestion by Fiat and Shamir [11]. In this model, called the random oracle model, concrete objects such that hash functions are treated as random objects. This allows to carry through

the usual reduction arguments to the context of relativized computations, where the hash function is treated as an oracle returning a random answer for each new query. A reduction still uses an adversary as a subroutine of a program that contradicts a mathematical assumption, such as the hardness of the DLP. However, probabilities are taken not only over coin tosses but also over the random oracle.

In the case of the discrete logarithm, whatever the computational model is, it is often the case that security is not related to the DLP itself, but rather to the so-called decisional Diffie-Hellman problem, or its computational version. The decisional Diffie-Hellman assumption over a group G asserts that it is hard to distinguish the distributions $\mathbf{D}$ and $\mathbf{R}$, where

$$\mathbf{R} = \{(g_1, g_2, u_1, u_2)\}$$

with all four elements taken at random in G and

$$\mathbf{D} = \{(g_1, g_2, u_1, u_2)\}$$

with $\log_{g_1}(u_1) = \log_{g_2}(u_2)$. A quantitative version measures the maximum advantage $\mathtt{AdvDDH}(\tau)$ of a statistical test T that runs in time τ. This means the maximum of the difference of the respective probabilities that T outputs 1, when probabilities are taken over $\mathbf{D}$ or $\mathbf{R}$.

Related to the above is the computational Diffie-Hellman assumption (CDH), which states that it is hard to compute g^{xy} from g, g^x, and g^y. It is obvious that DDH is a stronger assumption than CDH, which in turn, is stronger than the assumption that the DLP is hard. However, no other relation is known and the only way to solve the hard problems underlying DDH or CDH is to compute discrete logarithms. There are some indications, in other settings, that DDH might be easy, in some cases, while the CDH remains difficult (see [13] and [12]). However, the references just quoted are not relevant to the context of finite fields.

In conclusion, the only method known to attack the Diffie-Hellman problems is to solve the DLP. Therefore, in order to estimate whether the parameters of a cryptographic scheme, relying on the hardness of these problems, offer a wide security margin, one needs to refer to the performances of the various algorithms known for the DLP.

3 Algorithms for the DLP

3.1 Exponential algorithms

There is an algorithm due to Pohlig and Hellman (see [29]), which reduces the determination of the discrete logarithm in a field to the analogous problem in subgroups. For example, if $p-1$ is a product of small factors q_i, relatively prime to each other, the

algorithm separately operates on subgroups of order q_i However, this is irrelevant in a cryptographic context, since generation algorithms can simply avoid such primes p.

The best algorithm known to date for solving the DLP in any given group G is the Pollard ρ-method from [30] which takes computing time equivalent to about $\sqrt{\pi n/2}$ group operations. In 1993, van Oorschot and Wiener in [38], showed how the Pollard ρ-method can be parallelized so that, if t processors are used, then the expected number of steps by each processor before a discrete logarithm is obtained is $\simeq \frac{\sqrt{\pi n/2}}{t}$. In order to compute the discrete logarithm of y in base g, each processor computes a kind of random walk within elements of the form $g^a y^b$, selecting x_{i+1} as yx_i or x_i^2 or else gx_i, according to a deterministic but randomly looking choice (say based on a hash value). "Distinguished" points x_i are stored together with their representations $x_i = g^{a_i} y^{b_i}$ in a list that is common to all processors. When a collision occurs in the list, the requested discrete logarithm becomes known. There is no record involving discrete logarithms in subgroups of $\mathbb{Z}_p^\star$. However, one can estimate what such a record would be, by recalling the current record for computations in the group of points of an elliptic curve. In april 2000, the solution to the ECC2K-108 challenge from Certicom led to the computation of a discrete logarithm in a group with 2^{109} elements. This is one of the largest effort ever devoted to a public-key cryptography challenge. The amount of work required to solve the ECC2K-108 challenge was about 50 times that required to solve the 512-bit RSA cryptosystem (see [3]) and was thus close to 400000 mips-years. Because, the standard arithmetical operations execute faster than elliptic curve additions, an equivalent effort in the area of subgroups of $\mathbb{Z}_p^\star$ would presumably reach a few bits more. Referring to [26], we find that it would mean an extra 4 to 5 bits.

It should be noted that the exponential algorithms decribed in this section are superseded by the subexponential algorithms that appear further on in the present report. However, when dealing with subgroups, their computing time is related to the size of the subgroups rather than the size of the field. Accordingly, they give indications on the minimum size of the subgroup. Based on [26], it appears that moderate sizes such as 256 bits, should withstand attacks for the next 50 years.

3.2 The index calculus method

Efficient algorithms to solve the discrete logarithm problem over finite fields use the so-called index calculus method (see [27, 34] for a survey). In this section, we explain the basic principle of this method. Its latest instantiations, the number field sieve and the function field sieve, will be detailed later. We let $G = \mathbb{K}^*$ and we assume that the base g is a generator of G, so that all logarithms are well-defined. This is no restriction, since randomly chosen elements are generators with high probability.

The index calculus method uses a fixed small set called the *factor base* $\mathcal{B} \subseteq G$, and tries to write elements as a product of members of the factor base. The base consists

of objects which are small and irreducible, in an appropriate sense:

- In a prime field $\mathbb{F}_p$, where we identify field elements with integers in $\{0, \ldots, p-1\}$, a factor base consists of all prime numbers less than some prescribed bound.
- In a field of characteristic 2, $\mathbb{F}_{2^n}$, where we write field elements as polynomials of degree $< n$, a factor base consists of all irreducible polynomials of degree less than some prescribed bound.

The algorithm has two steps:

1. A precomputation step, where the logarithms $\log_g b$ of all members of the factor base are obtained
2. A computation step, which tries enough $g^a y$ until the result factors over the factor base, thus providing the requested logarithm $\log_g y$.

The first step itself has two stages:

- A sieving step, where one gathers multiplicative relations between elements of the factor base. Such relations are of the form $\prod_{b_i \in \mathcal{B}} b_i^{e_i} = 1$, where the product ranges over $\mathcal{B}$ and each e_i is an integer.
- A linear algebra step, where, taking logarithms, each relation produces a linear equation with unknowns the discrete logarithms of the factor base. When enough relations have been found, the logarithms of the factor base are obtained by linear inversion.

Observe that the various $\log_g b$, $b \in \mathcal{B}$, are computed up to some multiplicative constant. To find the value of the constant, one usually adds g as an element of $\mathcal{B}$. Also note that the first stage can easily be distributed.

Enlarging the factor base makes the sieving easier, but increases the number of relations needed to successfully perform the linear algebra. A fundamental problem in the analysis of index calculus algorithms is to estimate the probability that the sieving process produces enough relations. Such estimates rely on the heuristic assumption that elements appearing in the sieving process behave randomly. This assumption has been verified extensively. The resulting complexity estimates are naturally expressed in terms of the L-function:

$$L_q[s; c] = \exp(c(\ln q)^s (\ln \ln q)^{1-s}).$$

The reason why this function is involved is its relation with the asymptotic probability that a random element can be factored into elements of a factor base (see [18, 28]). For instance, it is known that the probability that a random integer $< L_x[\nu; \lambda]$ has all its prime factors $< L_x[w; \mu]$ is asymptotically

$$L_x[\nu - w; -\lambda(\nu - w)/\mu + o(1)].$$

3.3 The Number Field Sieve

In the case of prime fields, the method described in the previous section has been extended in [7], working with an imaginary quadratic number field. The extension has been termed *Gaussian Integer Method*. Its time complexity is $L_p[1/2; 1 + o(1)]$. It has been used until 1998 to establish records:

- 85 digits in 1996 (see [42])
- 90 digits in 1998 (see [19])

The Gaussian Integer Method has been generalized in [16, 32]. The resulting algorithm, called the *number field sieve*, adapts the general number field sieve (GNFS) factoring algorithm [25] to the computation of discrete logarithms in fields of small degree. It was first proposed by Gordon [16], in the case of prime fields $\mathbb{F}_p$, with a conjectured running time $L_p[1/3; 3^{2/3} + o(1)]$. The improved heuristic running time $L_p[1/3; (64/9)^{1/3} + o(1)]$ was obtained by Schirokauer [32]. Recently, Schirokauer [33] extended the number field sieve to any field of fixed degree. The complexity is still measured by the same function, replacing p by the cardinality p^n of the field. To keep the exposition simpler, we restrict oursekves to the case of a prime field $\mathbb{F}_p$: given $y \in \mathbb{F}_p^*$ and a generator g of $\mathbb{F}_p^*$, we wish to compute the unique $x \in \{1, \ldots, p-1\}$ such that

$$y \equiv g^x \pmod{p}.$$

3.3.1 Overview

The number field sieve (NFS) first selects two low-degree irreducible polynomials $f_1(X)$ and $f_2(X)$ in $\mathbb{Z}[X]$ with small coefficients such that $f_1(X)$ and $f_2(X)$ have a common root m in $\mathbb{F}_p$. These polynomials define two number fields $\mathbb{Q}(\alpha_1)$ and $\mathbb{Q}(\alpha_2)$. Because m is a root, there is a natural ring homomorphism φ_j from $\mathbb{Z}[\alpha_j]$ to $\mathbb{F}_p$ ($j \in \{1, 2\}$), induced by $\varphi_j(\alpha_j) = m$. We extend φ_j to the field $\mathbb{Q}(\alpha_j)$, ignoring potential divisibility problems.

The ring of integers of a number field is not necessarily a unique factorization domain, but the ring of fractional ideals always is. Thus, NFS selects two ideal factor bases $\mathcal{B}_1$ and $\mathcal{B}_2$ (corresponding to $\mathbb{Q}(\alpha_1)$ and $\mathbb{Q}(\alpha_2)$), consisting of prime ideals of smooth norm. We define other notions of smoothness as follows: a fractional ideal $\mathcal{I}$ of $\mathbb{Q}(\alpha_j)$ is *smooth* if it can be factored over the factor base $\mathcal{B}_j$, and an algebraic number x is *smooth* if the fractional ideal $\langle x \rangle$ it spans is smooth.

By sieving, the NFS finds a huge collection of pairs (a_i, b_i) of small integers such that each $a_i - b_i\alpha_j$ is smooth: as a result, the factorization of $\langle a_i - b_i\alpha_j \rangle$ is known. We first focus on the number field $\mathbb{Q}(\alpha_1)$. Linear algebra modulo $p - 1$ produces many

integer vectors (e_i) such that the fractional ideal

$$\prod_i \langle a_i - b_i\alpha_1\rangle^{e_i}$$

is a $(p-1)$-th power. This does not necessarily imply that the algebraic number $\prod_i (a_i - b_i\alpha_1)^{e_i}$ is a $(p-1)$-th power in $\mathbb{Q}(\alpha_1)$. However, using specific linear maps from $\mathbb{Q}(\alpha_1)$ to $\mathbb{Z}_{p-1}$, Schirokauer [32] was able to add a few linear equations to ensure that $\prod_i (a_i - b_i\alpha_1)^{e_i}$ is in fact a $(p-1)$-th power in $\mathbb{Q}(\alpha_1)$. If this holds, then:

$$\varphi_1\left(\prod_i (a_i - b_i\alpha_1)^{e_i}\right) \equiv 1 \pmod{p},$$

that is,

$$\prod_i (a_i - b_i m)^{e_i} \equiv 1 \pmod{p}.$$

Taking logarithms yields:

$$\sum_i e_i \log_g(a_i - b_i m) \equiv 0 \pmod{p-1}.$$

But each $a_i - b_i m$ is smooth because $a_i - b_i\alpha_j$ is smooth. Therefore the previous equation can be rewritten as:

$$\sum_{q\in\mathcal{B}} e'_q \log_g q \equiv 0 \pmod{p-1},$$

where $\mathcal{B}$ is a set of small prime numbers, and each e'_q is a known integer. Many such linear equations are obtained and more with the second number field $\mathbb{Q}(\alpha_2)$. When enough equations have been found, another use of linear algebra outputs $\log_g q$ for many $q \in \mathcal{B}$, up to some multiplicative constant. Such q's are called *good primes*. In our description, we mentioned using linear algebra twice, to produce $(p-1)$-th powers on one hand, and to compute the logarithms of good primes, on the other hand. They can actually be merged into a huge single step.

It remains to explain the computation of an individual logarithm. Note that applying the method to a single known power of g, will first remove the multiplicative constant. To compute the discrete logarithm x of $y = g^x \bmod p$, it is enough to find exponents $e_i \in \mathbb{Z}$ such that:

$$y \equiv \prod_{p_i\in\mathcal{B}} p_i^{e_i} \pmod{p},$$

where each p_i is a good prime, and not just $p_i \in \mathcal{B}$, since only the logarithms of the good primes are known. There are several methods to find such exponents. We

describe the technique presented in [23], which seems to be the most efficient known, as it does not require any huge linear algebra step contrary to earlier method of [40]. Using two-dimensional lattice reduction, one can compute two linearly independent integer vectors (A_1, B_1) and (A_2, B_2) with coordinates $\approx \sqrt{p}$ such that:

$$y \equiv \frac{A_1}{B_1} \equiv \frac{A_2}{B_2} \pmod{p}.$$

It follows that for any integers α and β:

$$y \equiv \frac{\alpha A_1 + \beta A_2}{\alpha B_1 + \beta B_2} \pmod{p}.$$

Now, since A_i and B_i are reasonably small, one can try to find pairs (α, β) such that both $\alpha A_1 + \beta A_2$ and $\alpha B_1 + \beta B_2$ are smooth with respect to good primes, using a sieving technique. If no candidate is found within a reasonable time, one tries again, replacing y by ys^i, where s is the largest good prime.

3.3.2 Practical experiments

The first practical experiment of NFS for discrete logarithms was presented at Eurocrypt '95 [39]. Further results appeared in [40, 41, 22]. The current record is [21], where discrete logarithms in a 120-digit prime field could be computed, following the approach of [23]. Paper [41] reports experiments with a larger 129-digit prime field, but its cardinality has a special form, which allows for better complexity. The current record of [21] was obtained in 10 weeks, on a single 525-MHz quadri-processor Digital Alpha Server 8400 computer. Sieving took 42 days and produced 2685597 equations with 1242551 unknowns. Structured Gaussian elimination [24, 23] took one extra day to reduce the system to 271654 equations in 271552 unknowns with 22690782 non-zero entries. Then, using a a parallel version of Lanczos's algorithm [14], linear algebra was performed in 30 days over 4 processors. One this has been done, the computation of each additional individual logarithm takes about 12 hours.

3.4 The Function Field Sieve

The function field sieve adapts the general number field sieve GNFS factoring algorithm [25] to the computation of discrete logarithms in finite fields of small characteristic. It was first discovered by Coppersmith [5] in the case of fields $\mathbb{F}_{2^n}$, with a conjectured running time $L_{2^n}[1/3; c + o(1)]$, where $c \approx 1.4$. Note that Coppersmith's algorithm predates the number field sieve. Adleman [1] proposed the function field sieve as a generalization of Coppersmith's algorithm to any finite fields of small characteristic. More precisely, the function field sieve computes discrete logarithms in $\mathbb{F}_{p^n}$

with expected running time $L_{p^n}[1/3; c + o(1)]$ for some constant c, when $\log p \leq n^{g(n)}$, where g is any function from $\mathbb{N}$ to $]0; 0.98[$ which converges to zero.

To keep the exposition simple, we only present Coppersmith's algorithm. This algorithm can be viewed as a special case of the function field sieve [1] (see [34]), though the constant in the complexity of the function field sieve is worse.

3.4.1 Overview of Coppersmith's algorithm

We want to compute discrete logarithms in the field $\mathbb{K} = \mathbb{F}_{2^n}$, which we represent, in the usual way, as $\mathbb{F}_2[X]/\langle f(X)\rangle$, where $f(X)$ is a monic irreducible polynomial of degree n over $\mathbb{F}_2$. Polynomial $f(X)$ is chosen of the form $X^n + f_1(X)$ where $f_1(X)$ has very small degree. It is conjectured, but not proven, that such an $f_1(X)$ with degree $O(\log n)$ always exists. Recall that squaring in $\mathbb{K}$ is a linear operation, which is computed for free and that factoring is easy.

We select a factor base $\mathcal{B}$ consisting of all irreducible polynomials of degree less than a prescribed bound, and define smoothness accordingly: a polynomial is *smooth* if it can be factored into elements of $\mathcal{B}$. Let r be such that $2^r \simeq n^{1/3}$ and $h = \lceil \frac{n}{2^r} \rceil$. Sieving, we search for polynomials $A(X)$ and $B(X)$ of degree approximately $n^{1/3}$, such that

$$C(X) = A(X)X^h + B(X),$$

and

$$D(X) = C(X)^{2^r} \equiv A^{2^r} X^{h2^r - n} f_1 + B^{2^r} \pmod{f(X)}.$$

are both smooth. The original sieving procedure of [5] was improved by [15]. Observe that the degrees of $C(X)$ and $D(X)$ are approximately $n^{2/3}$. When $C(X)$ and $D(X)$ are smooth, the relation $D(X) \equiv C(X)^{2^r} \pmod{f(X)}$ produces a linear equation modulo $2^n - 1$, with unknowns the logarithms of the members of the factor base. A huge linear algebra step yields all values of $\log b(X)$ for $b(X) \in \mathcal{B}$.

To compute the individual logarithm of y, one uses the fact that factoring can be efficiently performed in $\mathbb{K}$. One thus selects a random integer s until $yg^s \pmod{f(X)}$ is smooth, and then derives $\log_g y$.

3.4.2 Practical experiments

Experimental results are reported in [5, 15, 36]. The most recent record is [22], which computes logarithms over $\mathbb{F}_{2^{521}}$. The entire computation was done in one month on a single 525MHz quadri-processor Digital Alpha Server 8400 computer. The approach was based on a careful implementation of the general function field sieve [1], rather than the use of Coppersmith's algorithm. Sieving took 3 weeks and yielded 472121 equations with 450940 unknowns. Structured Gaussian elimination [24, 23] reduced the system to 197039 equations in 196939 unknowns with 12220108 non null entries, in

one hour. Then, it took 10 days to run Lanczos's algorithm [14] and complete the linear algebra. The computation of an individual logarithm takes an additional 12 hours.

Very recently, [36] presented a new implementation of Coppersmith's algorithm and preliminary experiments to compute discrete logarithms over $\mathbb{F}_{2^{607}}$. Sieving was completed and took 19,000 MIPS years. As a comparison, the factorization of RSA-155 required an estimated 8,000 MIPS years. Sieving yielded about $900,000$ relations involving about $760,000$ columns. After one day of structured Gaussian elimination, following [31]), the matrix was reduced to size $484,603 \times 484,603$. The linear algebra is still under way, using the improvement [37] of the block Wiedemann algorithm [6, 43].

Comparing both methods seems to suggest that the output of the sieving is better with the implementation [22] of the function field sieve, than with the implementation [36] of Coppersmith's algorithm. We mentioned that Coppersmith's algorithm could be viewed as a special case of the function field sieve. In Coppersmith's algorithm, one requires $C(X)$ and $C(X)^{2^r}$ to be simultaneously smooth. Without going into mathematical details at this point, we briefly mention that this corresponds to the choice of a polynomial of degree 2^r in the function field sieve. However, such degree 2^r may not be optimal compared to other choices, which are not a power of 2. Theoretical analysis actually suggests that the optimal degree should be $(n \log n)^{1/3}$. Since the function field sieve allows for any degree in selecting the polynomial, an appropriate implementation may be faster than Coppersmith's algorithm. Adleman already had mentioned in [1] that it should be possible to optimize and specialize the function field sieve in $\mathbb{F}_{2^n}$ and achieve a better running time than Coppersmith's algorithm. Besides, sieving tricks used in the number field sieve can be applied to the function field sieve, and not to Coppersmith's algorithm. It is therefore quite plausible that the method of [22] can be extended to larger fields, thus beating the performances of [36].

4 Consequences in terms of key sizes

4.1 The current status of the DLP

4.1.1 Prime fields

Surprisingly, it is very difficult to precisely estimate what is the largest size that current techniques for solving the DLP over prime fields could reach. The theoretical complexity $L_p[1/3; c + o(1)]$, $c \approx 1.92$, is a rough estimate, which is not of much use. Due to memory constraints, the size of the factor has to be kept significantly smaller than the optimal choice that theory would dictate.

This is in contrast with the situation of factoring. The largest factorization with the NFS factoring algorithm is a 512-bit RSA number [3] and this figure gives an accurate account of what factoring algorithms can achieve. Practical experience with the NFS method for the discrete logarithm appears more limited and the records are way behind.

The current record [21] only tackles a 120-digit prime field, which corresponds to bit-length 397. However, in [26], the authors state that

> It is generally accepted that, for any b in the current range of interest, factoring b-bit integers takes about the same amount of time as computing the discrete logarithm in $b - x$-bit field, where x is a small constant around 20.

This opinion may stem from the observation that the heuristic theoretical complexity of the NFS algorithms is the same for factoring and the DLP. Observe that the gap between records does not lead to a fair comparison: the computing power used in [21] is an order of magnitude below what [3] required. Still, it is conceivable that the gap is somehow larger than anticipated in [26]. Despite their similarities, there are several technical differences between the two NFS algorithms.

- The polynomial selection differs. For instance [23] used the fact that one can efficiently solve polynomial equations over a finite field, which has no equivalent in the setting of factoring. Different polynomial selections lead to different sieving regions and different choices of factor bases.
- The sieving also slightly differs. In the factorization setting, the linear algebra is over $\mathbb{F}_2$. This allows to use, besides the factor base, one or more so-called large primes.
- The linear algebra differs. In the factorization setting, the matrix is binary, whereas in the discrete logarithm setting, the matrix elements are integers modulo $p - 1$. This limits drastically the size of the factor base.

4.1.2 Fields of characteristic 2

It is equally difficult to precisely estimate what is the largest size that current techniques for solving the DLP over binary fields could reach. Again, the heuristic complexity $L_q[1/3; c + o(1)]$, where $c \approx 1.4$ for Coppersmith's algorithm, does not help much. Besides having lower complexity than NFS, Coppersmith's algorithm is much simpler to implement. Implementing the function field sieve is almost as difficult as implementing the number field sieve. However, because factoring polynomials over finite fields is easy, sieving can be done more efficiently in the function field sieve than in the number field sieve. Therefore, the function field sieve should be faster in practice than the number field sieve. Based on the above observations, it is no surprise that the two records in characteristic two are considerably larger than in the case of prime fields: $\mathbb{F}_{2^{607}}$ and $\mathbb{F}_{2^{521}}$. As noted in section 3.4.2, the former figure corresponds to a large scale experiment based on Copermsmith's algorithm, while the latter, based on the function field sieve, uses limited computing power, which leaves room for considerable improvement.

4.2 Selecting key sizes

In practical situations, the sizes of cryptographic keys based on the DLP have to be chosen so that they outreach the expected performances of the algorithms for solving the DLP, during the entire lifetime of the system. This involves making predictions. In [3], the authors derive the following formula

$$Y = 13.24D^{1/3} + 1928.6$$

for predicting the calendar year for factoring D-digit number by NFS. Following the opinion stated in [26], that the DLP in prime fields is 20 bits behind, which is apparently a conservative estimate, we obtain

$$Y = 13.24(D + 6)^{1/3} + 1928.6$$

for predicting the calendar year for the discrete logarithm. Applying the formula with $D = 309$, i.e. for a 1024 bit modulus, produces $Y = 2019$. Similarly, one gets $Y = 2042$ for 2048-bit integers and $Y = 2070$ for 4096-bit. Thus, current key sizes for cryptosystems appear safe. Although no similar formula has been proposed for fields of characteristic 2, it is advisable, based on the observations from section, to adopt key sizes that are much larger.

References

[1] L. M. Adleman. The function field sieve, *Proc. ANTS-I*, LNCS 877, Springer-Verlag, 108–121, 1994.

[2] M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols, *Proc. 1st CCS*, 62–73, ACM Press, New York, 1993.

[3] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of 512-bit RSA modulus, *Proc. Eurocrypt '00*, LNCS 1807, 1–18, Springer-Verlag, 2000.

[4] Certicom. Information on the Certicom ECC challenge, `http://www.certicom.com/research/ecc_challenge.html`

[5] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two *IEEE Trans. Inform. Theory*, 30(4):587–594, 1984.

[6] D. Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm, *Math. Comp.*, 62:333–350, 1994.

[7] D. Coppersmith, A.M. Odlyzko and R.Schroeppel. Discrete Logarithms in GF(p), *Algorithmica 1*, 1–15, 1986.

[8] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, *Proc. Crypto'98*, LNCS 1462, 13–25, 1998.

[9] Proposed federal Information Processing Standard for Digital Signature Standard, *Federal Register*, v.57, n.21, 3747–3749.

[10] T. El Gamal. A public key crtyptosystem and signature scheme based on discrete logarithms, *IEEE Trans. on Inform. theory*, 469–472, 1985.

[11] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions of Identification and Signature Problems, *Proc. Crypto '86*, LNCS 263, 186–194, Springer-Verlag, Berlin, 1987.

[12] G. Frey, M. Müller, and H. G. Rück. The Tate-Pairing and the Discrete Logarithm Applied to Elliptic Curve Cryptosystems, *IEEE Trans. Inform. Theory*, 45:1717–1719, 1999.

[13] G. Frey and H. G. Rück. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves, *Math. Comp.*, 62:865–874, 1994.

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

[15] D. Gordon and K. S. McCurley. Massively parallel computation of discrete logarithms, *Proc. Crypto '92*, LNCS 740, 312–323, Springer-Verlag, 1993.

[16] D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve, *SIAM J. Discrete Math.*, 6(1):124–138, 1993.

[17] R. Harley, D. Doligez, D. de Rauglaudre, X. Leroy, Elliptic Curve Discrete Logarithms: ECC2K-108,
`http://cristal.inria.fr/ harley/ecdl7/`

[18] A. Hildebrand and G. Tenenbaum. Integers without large prime factors, *J. Théor. Nombres Bordeaux*, (5):411–484, 1993.

[19] A. Joux and R. Lercier. Computing a discrete logarithm in $GF(p)$, p a 90 digit prime,
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[20] A. Joux and R. Lercier. Computing a discrete logarithm in $GF(p)$, p a 100 digit prime,
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[21] A. Joux and R. Lercier. Computing a discrete logarithm in $GF(p)$, p a 120 digits prime
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[22] A. Joux and R. Lercier. Computing a discrete logarithm in $GF(2^{521}$
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[23] A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields, *Math. Comp.*, 2001, to appear.

[24] B. A. Lamacchia and A. M. Odlyzko. Computation of discrete logarithm in prime fields, *Designs, Codes and Cryptography*, (1):47–62, 1991.

[25] A. K. Lenstra and H. W. Lenstra, Jr. *The Development of the Number Field Sieve*, Lecture Notes in Mathematics 1554, Springer-Verlag, 1993.

[26] A.K. Lenstra and E. Verheul. Selecting cryptographic key sizes, PKC'2000, LNCS 1751, 446–465, 2000.

[27] A. Odlyzko. Discrete logarithms: The past and the future, *Designs, Codes and Cryptography*, 19:129–145, 2000.

[28] D. Panario, X. Gourdon, and P. Flajolet. An analytic approach to smooth polynomials over finite fields, *Algorithmic Number Theory – Proc. ANTS-III*, LNCS 1423, Springer-Verlag, 1998.

[29] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Inform. Theory*, 24, 106–110, 1978.

[30] J. Pollard. Monte Carlo methods for index computation mod p, *Math. Comp.*, 32, 918–924, 1978.

[31] C. Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes, *Experiment. Math.*, 1(2):89–94, 1992.

[32] O. Schirokauer. Discrete logarithms and local units, *Philos. Trans. Roy. Soc. London Ser.*, A 345(1676):409–423, 1993.

[33] O. Schirokauer. Using number fields to compute logarithms in finite fields, *Math. Comp.*, 69(231):1627–283, 2000.

[34] O. Schirokauer, D. Weber, and T. Denny. Discrete logarithms: The effectiveness of the index calculus method, *Proc. ANTS-II*, LNCS 1122, Springer-Verlag, 1996.

[35] V. Shoup and T. Schweinberger. ACE Encrypt: The Advanced Cryptographic Engine' public key encryption scheme, Manuscript, March 2000. Revised, August 14, 2000.

[36] E. Thomé. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$, *Proc. Asiacrypt '01*, LNCS, to appear.

[37] E. Thomé. Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm, *Proc. ISSAC '01*. ACM Press, 2001.

[38] P.C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications, *J. Cryptology*, 12, (1999), 1–28.

[39] D. Weber. An implementation of the number field sieve to compute discrete logarithms mod p, *Proc. Eurocrypt '95*, LNCS 921, 95–105, Springer-Verlag, 1995.

[40] D. Weber. Computing discrete logarithms with the general number field sieve, *Proc. ANTS-II*, LNCS 1122, Springer-Verlag, 1996.

[41] D. Weber and T. Denny. The solution of McCurley's discrete log challenge, *Proc. Crypto '98*, LNCS 1462,458–471, Springer-Verlag, 1998.

[42] D. Weber, T. F. Denny and J. Zayer.
`http://felix.unife.it/Root/d-Mathematics/d-Number-theory`
`/t-Weber-discrete-logarithm-record-960925`

[43] D. H. Wiedemann. Solving sparse linear equations over finite fields, *IEEE Trans. Inform. Theory*, 32:54–62, 1986.