Oracle-Assisted Static Diffie-Hellman Is Easier Than Discrete Logarithms

Antoine Joux, Reynald Lercier, David Naccache, Emmanuel Thomé

Oracle-Assisted Static Diffie-Hellman Is Easier Than Discrete Logarithms

Antoine Joux[1], Reynald Lercier[2], David Naccache[3], and Emmanuel Thomé[4]

[1] DGA and Université de Versailles, UVSQ PRISM 45 avenue des États-Unis
F-78035 Versailles CEDEX, France
antoine.joux@m4x.org
[2] DGA/CELAR, La Roche Marguerite, F-35174 Bruz, France and
IRMAR, Université de Rennes 1, Campus de Beaulieu, F-35042 Rennes, France
reynald.lercier@m4x.org
[3] École normale supérieure, Équipe de cryptographie, 45 rue d'Ulm
F-75230 Paris CEDEX 05, France
david.naccache@ens.fr
[4] INRIA Lorraine, LORIA, CACAO – bâtiment A, 615 rue du Jardin botanique
F-54602 Villiers-lès-Nancy CEDEX, France
emmanuel.thome@normalesup.org

Abstract. This paper extends Joux-Naccache-Thomé's e-th root algorithm to the static Diffie-Hellman problem (SDHP).

The new algorithm can be adapted to diverse finite fields by customizing it with an NFS-like core or an FFS-like core.

In both cases, after a number of non-adaptive SDHP oracle queries, the attacker builds-up the ability to solve new SDHP instances *unknown before the query phase*.

While sub-exponential, the algorithm is still significantly faster than all currently known DLP and SDHP resolution methods.

We explore the applicability of the technique to various cryptosystems.

The attacks were implemented in $\mathbb{F}_{2^{1025}}$ and also in $\mathbb{F}_p$, for a 516-bit p.

Keywords: DLP, SDHP, FFS, NFS.

1 Introduction

In [11], Joux, Naccache and Thomé showed that carefully interacting with an e-th root oracle improves one's ability to compute $\sqrt[e]{t} \bmod n$ for arbitrary t *after* the communication with the oracle.

In Joux *et alii*'s game, the attacker can only query the oracle during a learning phase that takes place *before* t is known. As this learning phase ends, the attacker receives the target t and outputs it's e-th root. In other words, [11] establishes that the release of *certain modular roots* leaks information that eases the calculation of all *future* roots.

Work partially supported by DGA research grant 05.34.058

This is interesting because, while sub-exponential and oracle-assisted, [11] is still faster than GNFS-factoring.

A recent ePrint publication ([12]) extended [11] to discrete logarithms (DLs). While complexities and details vary, [12]'s overall scenario remains essentially the same, replacing root extractions by DLP calculations.

This work tackles the static Diffie-Hellman problem (SDHP) on which many cryptosystems rely (*e.g.*, [2, 4, 5] – to mention only a few). Namely, we show how after several carefully crafted non-adaptive SDHP oracle queries, an attacker can improve his ability to solve arbitrary SDHP instances, *unknown before the last oracle query*. The attacker's workload is significantly lower than [12][5] and all currently known DLP and SDHP resolution algorithms.

The method can be adapted to diverse finite fields by customizing it either with an NFS-like core or an FFS-like core.

We assume that the reader is familiar with [1, 11, 12], whose notations, introductory descriptions and terminology we extensively borrow and recycle.

Results: Let $Q = q^n$ where q is a prime power[6] and n an integer.

We present an algorithm $\mathcal{A}$ performing, during a learning phase, L non adaptive SDHP-oracle queries $\{x_1, \ldots, x_L\} \in \mathbb{F}_Q^L$ to which the oracle responds with $\{x_1^d, \ldots, x_L^d\} \in \mathbb{F}_Q^L$ for some secret $d \in [0, Q-1]$.

After the learning phase, $\mathcal{A}$ is given a *previously unseen* challenge z and outputs $z^d \in \mathbb{F}_Q$.

The $L_Q(\frac{1}{3}, \sqrt[3]{x})$-complexities of our algorithms are expressed in the following table:

variant	oracle accesses	learning phase time	post-learning phase time
FFS	$\frac{4}{9}$	-	$\frac{4}{9}$
NFS-HD	$\frac{48}{91}$	$\frac{384}{91}$	$\frac{384}{91}$
NFS	$\frac{4}{9}$	$\frac{32}{9}$	3

Here, NFS-HD stands for high-degree NFS to emphasize the difference between this middle case and the regular NFS.

Let us recall that x values of the FFS, NFS-HD and NFS are respectively $\frac{32}{9}$, $\frac{128}{9}$ and $\frac{64}{9}$.

The following sections will detail $\mathcal{A}$'s inner mechanics[7], complexity and the relation between L and $\{q, n\}$.

The attacks was implemented in $\mathbb{F}_{2^{1025}}$ and also in $\mathbb{F}_p$, for a 516-bit p (details in the Appendix).

[5] Note, however, that [12] addresses a different problem than ours.

[6] For simplicity, we assume that q is prime although for some composite degree extensions it is sometime more efficient to choose a composite subfield as a base field.

[7] Note that as the learning phase takes place before z is disclosed, $\{x_1, \ldots, x_L\}$ and L *only* depend on Q.

2 Conventions

Complexity: We express complexities using the following notations[8]:

$$L_Q(\nu,\lambda) = \exp\left(\lambda\,(1+o(1))\,(\log Q)^{\nu}\,(\log\log Q)^{1-\nu}\right),$$
$$\ell_Q(\nu) = (\log Q)^{\nu}\,(\log\log Q)^{1-\nu} \quad \text{and} \quad D_Q(\nu,\lambda) = \lfloor \log_q(L_Q(\nu,\lambda))\rfloor.$$

As $L_Q(\nu,\lambda)$ includes an $o(1)$, $D_Q(\nu,\lambda)$ inherently stands for $D_Q(\nu,\lambda+o(1))$.

Following [9, 10], we guarantee that the complexity of our algorithms is $L(\frac{1}{3},O(1))$ by distinguishing the following three cases:

FFS: $\log q \in o(\ell_Q(\frac{1}{3}))$

▶ q is asymptotically smaller than all functions in $L_Q(\frac{1}{3},\gamma)$.

NFS-HD: $\log q \in [\omega(\ell_Q(\frac{1}{3})), o(\ell_Q(\frac{2}{3}))]$

▶ q is asymptotically between all functions in $L_Q(\frac{1}{3},\gamma)$ and $L_Q(\frac{2}{3},\gamma)$.

NFS: $\log q \in \omega(\ell_Q(\frac{2}{3}))$

▶ q is asymptotically greater than all functions in $L_Q(\frac{2}{3},\gamma)$.

Assumptions: We rely on the usual heuristic assumptions [3, 14, 13]:

- The probability that an integer $\mathfrak{q} \sim L_Q(\nu_\mathfrak{q},\lambda_\mathfrak{q}+o(1))$ is a product of prime factors of magnitude at most $L_Q(\nu_\mathfrak{p},\lambda_\mathfrak{p}+o(1))$ is:
$$L_Q\left(\nu_\mathfrak{q}-\nu_\mathfrak{p}, -\frac{\lambda_\mathfrak{q}}{\lambda_\mathfrak{p}}(\nu_\mathfrak{q}-\nu_\mathfrak{p})+o(1)\right)$$
despite the fact that integers $\mathfrak{q}$ might follow non-uniform distributions.
- The probability that a polynomial $\mathfrak{q} \in \mathbb{F}_q[X]$ of degree $D_Q(\nu_\mathfrak{q},\lambda_\mathfrak{q}) \geqslant 1$ is a product of irreducible polynomials of degree at most $D_Q(\nu_\mathfrak{p},\lambda_\mathfrak{p}) \geqslant 1$ is also:
$$L_Q\left(\nu_\mathfrak{q}-\nu_\mathfrak{p}, -\frac{\lambda_\mathfrak{q}}{\lambda_\mathfrak{p}}(\nu_\mathfrak{q}-\nu_\mathfrak{p})+o(1)\right)$$

3 Function Field Sieve Oracle-Assisted SDH Resolution

3.1 The Algorithm

In this section, we denote by ε the constant $\varepsilon = \sqrt[3]{\frac{4}{9}}$ and by $\vartheta_{n+1} = \frac{1}{3}(1+\frac{1}{2^n})$.

[8] Throughout this paper log denotes the natural logarithm.

Polynomial Construction. Let $D = D_Q(\vartheta_{n+1}, \varepsilon)$, D tends to infinity with Q because $\log q \in o(\ell_Q(\frac{1}{3}))$ and thus $D \geqslant 1$.

Let $d_1 \cong \sqrt{nD}$ and $d_2 \cong \sqrt{\frac{n}{D}}$ with $d_1 d_2 \geqslant n$.

Choose two univariate polynomials $f_1, f_2 \in \mathbb{F}_q[x]$ of respective degrees d_1, d_2, such that the resultant of $F_1(x,y) = y - f_1(x)$ and $F_2(x,y) = x - f_2(y)$ with respect to variable y has an irreducible factor of degree n over $\mathbb{F}_q$.

Let $f(x)$ denote this irreducible factor of $\mathrm{Res}_y(F_1, F_2) = x - f_2(f_1(x))$, then clearly $\mathbb{F}_{q^n} \simeq \mathbb{F}_q[x]/(f(x))$. Let α be a root of $f(x)$ in $\mathbb{F}_{q^n}$ and let $\beta = f_1(\alpha)$. Then, by construction, we have $\alpha = f_2(\beta)$ too.

Learning Phase. We let $\mathcal{F}_D$ be the set of all monic irreducible polynomials over $\mathbb{F}_q$ of degree up to D and define $\{x_i\} = \{\{\mathfrak{p}(\alpha), \mathfrak{p}(\beta)\} \mid \mathfrak{p}(x) \in \mathcal{F}_D\}$. After about $2q^D \simeq L_Q(\frac{1}{3}, \varepsilon + o(1))$ SDHP-oracle queries, we get $\{\mathfrak{p}(\alpha)^d, \mathfrak{p}(\beta)^d\}_{\mathfrak{p} \in \mathcal{F}_D}$.

Descent. The descent phase consists in finding a multiplicative relation between the target z and the elements of $\mathcal{F}_\alpha \cup \mathcal{F}_\beta$. To do so, we use a special-$\mathfrak{q}$ sieving subroutine (hereafter SqSS) as a basic building block. SqSS is called iteratively to steadily decrease the degree of intermediate polynomials.

The SqSS works as follows:

Let $\mathfrak{q}$ be an irreducible polynomial in x (respectively y) of degree $d_\mathfrak{q}$.

To ease SqSS's analysis we write $d_\mathfrak{q}$ as:

$$d_\mathfrak{q} = D_Q(\nu_\mathfrak{q}, \lambda_\mathfrak{q}) = (\lambda_\mathfrak{q} + o(1))\, n^{\nu_\mathfrak{q}} \log_q^{1-\nu_\mathfrak{q}} n$$

for two constants $\nu_\mathfrak{q}, \lambda_\mathfrak{q}$ such that $\sqrt[3]{n} \leqslant d_\mathfrak{q} \leqslant n$. Note that the values $\nu_\mathfrak{q}, \lambda_\mathfrak{q}$ *evolve simultaneously* – as will be seen later in further detail.

SqSS's purpose is to write an element $\mathfrak{q}(\alpha)$ or $\mathfrak{q}(\beta)$ as a product of elements of the form $\mathfrak{p}(\alpha)$ and $\mathfrak{p}(\beta)$, where the $\mathfrak{p}$-polynomials are of degrees smaller than the degree of $\mathfrak{q}$.

Let $d_\varepsilon = D_Q(\frac{1}{3}, \varepsilon)$, we consider $q^{d_\varepsilon} \simeq L_Q(\frac{1}{3}, \varepsilon)$ polynomials:

$$H(x,y) = \sum_{\substack{0 \leq i \leq d_x \\ 0 \leq j \leq d_y}} h_{i,j} x^i y^j \in \mathbb{F}_q[x,y]$$

$$\text{where } \; d_y = \left\lceil \max(\sqrt{(d_\mathfrak{q} + d_\varepsilon)/D} - 1, 1) \right\rfloor \; \text{ and } \; d_x = \left\lceil \frac{d_\mathfrak{q} + d_\varepsilon + 1}{d_y + 1} \right\rceil - 1$$

We only consider polynomials $H(x,y)$ such that $\mathfrak{q}$ divides $H_x = \sum_{i,j} h_{i,j} x^i f_1(x)^j$ (respectively $H_y = \sum_{i,j} h_{i,j} f_2(y)^i y^j$).

The complexity analysis (*cf. infra*) shows that there exists, amongst these polynomials, polynomials H such that H_x and H_y both admit a factorization of the form:

$$\prod_{\substack{\mathfrak{p} \in \mathcal{F}_{D_Q(\nu_\mathfrak{p},\lambda_\mathfrak{p})} \\ \text{for } \mathfrak{p} \neq \mathfrak{q}}} \mathfrak{p} \quad \text{with} \quad \begin{cases} \nu_\mathfrak{p} = \dfrac{1}{6} + \dfrac{\nu_\mathfrak{q}}{2} & \text{and } \lambda_\mathfrak{p} = \dfrac{2\sqrt{\lambda_\mathfrak{q}+\varepsilon}}{3\varepsilon} \quad \text{if } \nu_\mathfrak{q} > \vartheta_{n+1} \\ \nu_\mathfrak{p} = \nu_\mathfrak{q} & \text{and } \lambda_\mathfrak{p} = \dfrac{3\varepsilon+\lambda_\mathfrak{q}}{6\varepsilon\sqrt{\varepsilon}} \quad \text{otherwise.} \end{cases}$$

By proceeding so, we can express an element $\mathfrak{q}(\alpha)$ or $\mathfrak{q}(\beta)$ as a product of polynomials $\mathfrak{p}$ of degrees at most $D_Q(\nu_\mathfrak{p}, \lambda_\mathfrak{p})$ evaluated at α or β.

Descending Using SqSS. Define the sequences:

$$\nu_{k+1} = \frac{1}{6} + \frac{\nu_k}{2} \quad \text{and} \quad \lambda_{k+1} = \frac{2\sqrt{\lambda_k + \varepsilon}}{3\varepsilon} \quad \text{with} \quad \nu_0 = \lambda_0 = 1$$

We note that, indeed, $\nu_k = \vartheta_k$ and that the limit at infinity of the sequence λ_k is equal to $\frac{1+\sqrt{5}}{\sqrt[3]{18}} \cong 1.234$. As our procedures perform a finite number of steps, these *ad infinitum* convergence targets are only approached. However, the limit at infinity guarantees that during the phase of the descent where ν decreases, λ remains bounded (smaller than the above limit). After that point ν remains constant and λ quickly tends to ε.

The descent starts with one polynomial $\mathfrak{q}(x)$ of maximal degree $D_Q(\nu_0, \lambda_0)$ with $z = \mathfrak{q}(\alpha)$. After a first invocation of SqSS we get polynomials $\mathfrak{p}$ in x and y of degree at most $D_Q(\nu_1, \lambda_1)$ where $\nu_1 = \frac{2}{3}$ and $\lambda_1 = \sqrt{\varepsilon(1+\varepsilon)}$.

Again, we hand over each intermediate polynomial $\mathfrak{p}$ to SqSS to obtain new polynomials $\mathfrak{p}$ in x and y of degree at most $D_Q(\nu_2, \lambda_2)$ *etc.*

After $N = O(\log n)$ iterations involving intermediate polynomials of degrees $D_Q(\nu_k, \lambda_k)$, we eventually reach polynomials of degree at most $D_Q(\vartheta_{n+1}, \lambda_N)$.

Consider the sequence $\lambda'_{k+1} = \frac{3\varepsilon+\lambda'_k}{6\varepsilon\sqrt{\varepsilon}}$ starting at $\lambda'_0 = \lambda_N = \frac{1+\sqrt{5}}{\sqrt[3]{18}} + o(1)$.

We iterate SqSS $O(\log n)$ additional times; thereby generating polynomials of degree $D_Q(\vartheta_{n+1}, \lambda'_k)$. Eventually, we reach polynomials of degree at most D, *i.e.* $D_Q(\vartheta_{n+1}, \varepsilon)$.

It remains to combine these relations to write z as a fraction of products of elements of the form $\mathfrak{p}(\alpha)$ or $\mathfrak{p}(\beta)$ with the degrees of $\mathfrak{p}$ bounded by D. Finally, we use the oracle's answer-list $\{\mathfrak{p}(\alpha)^d, \mathfrak{p}(\beta)^d\}_{\mathfrak{p} \in \mathcal{F}_D}$, to retrieve z^d.

3.2 Complexity

We analyze the complexity of each step:

SqSS For $\nu_\mathfrak{q} > \vartheta_{n+1}$: The sum of the degrees of H_x and H_y is:

$$(d_x + 1)\sqrt{(d_\mathfrak{q} + d_\varepsilon)/D} + (d_y + 1)\sqrt{(d_\mathfrak{q} + d_\varepsilon)D} - 2 \simeq 2\sqrt{(d_\mathfrak{q} + d_\varepsilon)n}$$

As, in addition, $\mathfrak{q}$ divides one of these two polynomials, our smoothness probability is identical to that of $d_\mathfrak{p}$-smooth polynomials of degree $2\sqrt{(d_\mathfrak{q} + d_\varepsilon)n} - d_\mathfrak{q}$ *i.e.*:

$$2\sqrt{(d_{\mathfrak{q}}+d_\varepsilon)n}-d_{\mathfrak{q}} \leqslant 2(\sqrt{\lambda_{\mathfrak{q}}+\varepsilon}+o(1))n^{\frac{1+\nu_{\mathfrak{q}}}{2}}\log_q^{\frac{1-\nu_{\mathfrak{q}}}{2}} n \leqslant D_Q\left(\frac{1+\nu_{\mathfrak{q}}}{2}, 2\sqrt{\lambda_{\mathfrak{q}}+\varepsilon}\right)$$

And the probability that polynomials of this degree are $D_Q(\frac{1}{6}+\frac{\nu_{\mathfrak{q}}}{2}, \frac{2\sqrt{\lambda_{\mathfrak{q}}+\varepsilon}}{3\varepsilon})$-smooth is $L_Q(\frac{1}{3}, -\varepsilon+o(1))$ (*cf.* to section 2).

SqSS For $\nu_{\mathfrak{q}} \leqslant \vartheta_{n+1}$: The sum of the degrees of H_x and H_y is:

$$\sqrt{nD}+\frac{d_{\mathfrak{q}}+d_\varepsilon}{2}+\frac{d_{\mathfrak{q}}+d_\varepsilon}{2}\sqrt{\frac{n}{D}}+1 \simeq \sqrt{nD}+\frac{d_{\mathfrak{q}}+d_\varepsilon}{2}\sqrt{\frac{n}{D}}$$

Then,

$$\begin{aligned}\sqrt{nD}+\frac{d_{\mathfrak{q}}+d_\varepsilon}{2}\sqrt{\frac{n}{D}}-d_{\mathfrak{q}} &\leqslant \left(\sqrt{\varepsilon}+\frac{\lambda_{\mathfrak{q}}}{2\sqrt{\varepsilon}}+\frac{\sqrt{\varepsilon}}{2}+o(1)\right) n^{\frac{2}{3}+\frac{1}{3\,2^n}}\log_q^{\frac{2}{3}-\frac{1}{3\,2^n}} n \\ &\leqslant D_Q\left(\frac{2}{3}+\frac{1}{3\,2^n}, \frac{3\varepsilon+\lambda_{\mathfrak{q}}}{2\sqrt{\varepsilon}}\right)\end{aligned}$$

Again, the odds that polynomials of this degree are $D_Q(\vartheta_{n+1}, \frac{3\varepsilon+\lambda_{\mathfrak{q}}}{6\varepsilon\sqrt{\varepsilon}})$-smooth is $L_Q(\frac{1}{3}, -\varepsilon+o(1))$.

In both cases, after enumerating $L_Q(\frac{1}{3}, \varepsilon+o(1))$ polynomials, we expect to find a good polynomial H.

Descent: SqSS calls involve medium-sized $\mathfrak{p}$-polynomials of degree larger than D. The number of these polynomials is thus bounded by $O(\frac{n}{D})$, *i.e.* $O(\sqrt[3]{n^2\log_q^2 n})$.

As the whole process involves $O(\log n)$ SqSS calls, the total number of medium-size polynomials $\mathfrak{p}$ to be considered is:

$$O\left(\frac{n}{\log_q n}\right)^{\frac{2}{3}O(\log n)} = e^{O(\log^2 n)}$$

Running SqSS on each of these polynomials, costs $L_Q(\frac{1}{3}, \varepsilon+o(1))$, which is far larger than $\exp(O(\log^2 n))$; we thus conclude that the descent's complexity is $L_Q(\frac{1}{3}, \varepsilon+o(1))$, having pre-computed a table of $L_Q(\vartheta_{n+1}, \varepsilon+o(1))$ elements $\{\mathfrak{p}_i(\alpha)^d\}_i \cup \{\mathfrak{p}_i(\beta)^d\}_i$.

Since $\log^{\pm 2^{-n}} q^n = 1+o(1)$

$$\begin{aligned}L_Q(\vartheta_{n+1}, \varepsilon+o(1)) &\simeq \exp\left((\varepsilon+o(1))(\log^{\frac{1}{3}+\frac{1}{3\,2^n}} q^n)(\log\log^{\frac{2}{3}-\frac{1}{3\,2^n}} q^n)\right) \\ &\simeq L_Q(\frac{1}{3}, \varepsilon+o(1))\end{aligned}$$

Total Complexity: The attack's total complexity is hence $L_Q(\frac{1}{3}, \varepsilon + o(1)) = L_Q(\frac{1}{3}, \sqrt[3]{\frac{4}{9}} + o(1))$, in time, space and oracle accesses.

4 Number Field Sieve Oracle-Assisted SDH Resolution

4.1 The Algorithm

Due to lack of space, we only sketch the two NFS algorithms covering large and medium base-field cardinality-values q. As our algorithms preserve, *mutatis mutandis*, most FFS features, we will mainly focus on the specificities proper to these two cases. We also refer the reader to [11], whose arbitrary e-th roots procedure has to be slightly modified to fit the case under consideration.

Algebraic Side: In the FFS case, thanks to the existence of an *ad-hoc* construction, both sides are rational, *i.e.* on both side we only encounter polynomials. With the NFS matters get more complicated. Indeed, all known polynomial constructions for NFS procedures involve at least one non-trivial number field.

For example over $\mathbb{F}_p$ the base-m construction yields two polynomials $f_1(x) = x - m$ and $f_2(x)$, such that $p \mid f_2(m)$. Since f_1 is a linear polynomial, it generates a trivial number field: the corresponding side is rational and can be addressed using integers. However, f_2 defines a number field $\mathbb{Q}(\alpha)$ with non trivial unit and class groups.

Consequently, to deal with f_2's side[9], we need to work with factorizations into ideals: the *algebraic factor base* consists of ideals of small norm. This change truly complicates the approach. As the ideals in the factor base are usually non-principal, we do not know how to construct an oracle query corresponding to a given ideal.

Instead, we need to find smooth $a - b\alpha$ values which factor into small ideals and "translate" them to integers $a - bm$ for the oracle.

Linear Algebra: To relate these factorizations back to multiplicative relations in $\mathbb{F}_Q$ we must resort to linear algebra, which can be done in two ways:

- Repeating the descent of [11], it is possible (using the oracle's answers) to relate the challenge $z \in \mathbb{F}_Q$ to some $a - bm \in \mathbb{Z}$ such that $a - b\alpha$ factors into small ideals. By solving a linear system over $\mathbb{Z}/(Q-1)\mathbb{Z}$, this exact factorization can be mapped to a multiplicative combination of oracle answers.

  Note that to transform the equality of ideal factorizations into an equality that makes sense in $\mathbb{F}_Q$, one must include additive characters in $\mathbb{Z}/(Q-1)\mathbb{Z}$, *à la* Schirokauer [15]. Since this methodology performs a linear algebra step that depends on the challenge, it therefore yields an algorithm where the linear algebra step has to be redone for each challenge.

[9] Called the *algebraic side*.

– There is also a second method – very specific to our setting. It is possible to use linear algebra to compute virtual SDH values that correspond to each ideal[10]. From a theoretical standpoint, this can be justified as in [8, 16]. Note that this second method also requires the inclusion of additive characters *à la* Schirokauer.

Note that in some cases (NFS-HD), there are two algebraic sides to consider.

We start from a large system of equations, where each equation expresses a value in $\mathbb{F}_Q$ – the equation's generator – as a product of ideals in the number field. Moreover, we are also given the power of this value returned by the oracle. Our goal is to associate to each ideal I a number P_I such that in each equation, the substitution of I by P_I will yield a product whose value matches the oracle's answer for this equation.

If we can achieve this goal, P_I can be looked upon as an oracle output for I, and we are back in a FFS-like scenario. However, this is not completely straightforward:

A first (costly) option would be to use linear algebra over the integers and express each ideal as a product of generators (possibly with rational exponents) but one may note that the vector of generators and the vector[11] of ideals are related by a matrix multiplication in the exponent. More precisely, if ideals are indexed from 1 to B, and denoting by G_i the i-th equation generator ($1 \leq i \leq B$), there exists a $B \times B$ matrix $\mathbf{A}$, such that:

$$\begin{pmatrix} G_1 \\ G_2 \\ \vdots \\ G_B \end{pmatrix} = \mathbf{A} \star \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_B \end{pmatrix} \tag{1}$$

where $\star$ denotes matrix multiplication in the exponent, *i.e.* where for all i:

$$G_i = \prod_{j=1}^{B} I_j^{\mathbf{A}_{i,j}} \tag{2}$$

Moreover this matrix multiplication in the exponent is defined modulo $Q-1$, the cardinality of $\mathbb{F}_Q^*$. Having observed that, we can use Wiedemann's algorithm [17] to obtain the P_I values. *e.g.* if $\mathbf{A}$ is invertible, Wiedemann's algorithm finds an expression $\mathbf{A}^{-1} = F(\mathbf{A})$ where F is a polynomial.

From this expression, we obtain:

$$\begin{pmatrix} P_{I_1} \\ P_{I_2} \\ \vdots \\ P_{I_B} \end{pmatrix} = F(\mathbf{A}) \star \begin{pmatrix} \mathcal{O}(G_1) \\ \mathcal{O}(G_2) \\ \vdots \\ \mathcal{O}(G_B) \end{pmatrix} \tag{3}$$

[10] We assume here that the class number of $\mathbb{Q}(\alpha)$ is co-prime with the relevant prime factors of $Q-1$. This assumption is clearly expected to hold.

[11] In this vector, there are also a few additional components corresponding to the additive characters.

where $\mathcal{O}(G_i)$ denotes the oracle's output for the generator G_i.

This linear algebra step based on [17] has essentially the same complexity as usual, with the small caveat that additions are replaced by multiplications and multiplications by constants are replaced by exponentiations.

4.2 Complexity

The $L_Q(\frac{1}{3}, \sqrt[3]{x})$-complexities of the NFS and NFS-HD oracle-assisted attacks are expressed in the following table:

variant	oracle accesses	learning phase time	post-learning phase time
NFS-HD	$\frac{48}{91}$	$\frac{384}{91}$	$\frac{384}{91}$
NFS	$\frac{4}{9}$	$\frac{32}{9}$	3

These figures stem from the technical analysis found in the Appendix.

5 Cryptanalytic Applications

This section follows very closely Brown and Gallant's [1]. Variable were renamed to ease the identification of on parameters playing specific roles in our algorithms (*e.g.* secret d, oracle queries x_i *etc*).

It is clear that any cryptographic protocol where the secret-owner plays the role of a SDHP-oracle will succumb to the new attacks. The technique can apply to primitives as diverse as public-key encryption, digital signature and key retrieval.

5.1 Textbook El-Gamal Encryption

As observed in [1], textbook El-Gamal encryption makes an SDHP oracle available. El-Gamal encryption of message m to a receiver with public-key g^d results in a ciphertext $c = \{c_1, c_2\} = \{g^r, m \cdot g^{dr}\}$.

In a chosen ciphertext attack against an encryption scheme, an adversary $\mathcal{A}$ can select any ciphertext and obtain its decryption. For El-Gamal encryption, if $\mathcal{A}$ chooses $c = \{x, c_2\}$, then he obtains $m = c_2/x^d$. $\mathcal{A}$ can thus obtain $x^d = c_2/m$, thereby transforming the receiver into an SDHP-oracle for the challenge x.

Textbook El-Gamal is already known to be vulnerable to chosen ciphertext attacks. The known attacks mainly allow information to be learned about previously encrypted messages but do not leak anything about d. In the scenario described here, an attacker use the receiver as an oracle to learn how to decrypt future traffic sub-exponentially, but still significantly faster than all previously known SDHP-solving algorithms.

5.2 Ford-Kaliski Key Retrieval

In Ford-Kaliski's key retrieval protocol [5], a client picks a random exponent b and computes $x = g^b$.

x is sent to the server who, replies with $s = x^d$. The client computes $t = \sqrt[b]{s}$ and retrieves the *hardened password* $t = \sqrt[b]{s} = g^d$.

The protocol's purpose is to transform a low-entropy secret password g into a hardened password g^d involving a longer secret known only to the server. This prevents dictionary attacks on the client's password.

Therefore, the server plays exactly the role of the SDHP oracle considered in this paper.

Variants of this protocol are currently under discussion in two standardization groups [6, 7].

5.3 Chaum- Van Antwerpen's Undeniable Signatures

Another protocol in which an SDHP oracle is available is Chaum and van Antwerpen's undeniable signature scheme [2]. The protocol uses a prime modulus $p = 2q + 1$ such that q is prime.

Let $g \in \mathbb{Z}_p^*$ be an element of order q and $y = g^d$ where d is the signer's private key (all other parameters are public). To sign a message x, Alice produces the signature $s = x^d \bmod p$. Bob verifies s interactively:

Bob picks $\{e_1, e_2\}$ at random in $\mathbb{Z}_q$, and sends $c = s^{e_1} y^{e_2}$ to Alice.

Alice computes $h = \sqrt[d]{c} \bmod p$ and sends it to Bob. Bob accepts s as a valid signature if $h = x^{e_1} g^{e_2} \bmod p$.

Note that [2], provides SDHP oracles for both d and $\frac{1}{d} \bmod q$.

References

1. D. Brown and R. Gallant, *The static Diffie-Hellman problem*, `eprint.iacr.org`, Cryptology ePrint Archive, Report 2004/306, 2004.
2. D. Chaum and H. van Antwerpen, *Undeniable signatures*, Proceedings of CRYPTO 1989, LNCS, vol. 435, Springer-Verlag, pp. 212–217, 1989.
3. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Ark. Mat. Astr. Fys. 22, pp. 1–14, 1930.
4. T. El-Gamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, Proceedings of CRYPTO 1984, LNCS, vol. 196, Springer-Verlag, pp. 10–18, 1985.
5. W. Ford and B. Kaliski, *Server-assisted generation of a strong secret from a password*, Nineth international workshop on enabling technologies - WET ICE 2000, IEEE Press, 2000.
6. IEEE P1363.2/D23, *Draft standard for specifications for password-based public key cryptographic techniques*, p. 24 and on, March 2006.
7. ISO 11770-4, *Information technology - security techniques - key management - part 4: Mechanisms based on weak secrets*, ISO, November 2004.

8. A. Joux and R. Lercier, *Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method*, Mathematics of computation, vol. 242(72), pp. 953–967, 2003.
9. A. Joux and R. Lercier, *The function field sieve in the medium prime case*, Proceedings of EUROCRYPT 2006, LNCS, vol. 4004, Springer-Verlag, pp. 254–270, 2006.
10. A. Joux, R. Lercier, N. Smart and F. Vercauteren, *The number field sieve in the medium prime case*, Proceedings of CRYPTO 2006, LNCS, vol. 4117, Springer-Verlag, pp. 326–344, 2006.
11. A. Joux, D. Naccache and E. Thomé, *When e-th roots become easier than factoring*, Proceeding of ASIACRYPT 2007, LNCS, vol. 4833, Springer-Verlag, pp. 13–28, 2007.
12. N. Koblitz and A. Menezes, *Another look at non-standard discrete log and Diffie-Hellman problems*, `eprint.iacr.org`, Cryptology ePrint Archive, Report 2007/442, 2007.
13. A. Lenstra, H. Lenstra, M. Manasse and J. Pollard *The number field sieve*, In A. Lenstra and H. Lenstra Eds., *The development of the number field sieve*, LNM vol. 1554, pp. 11–42. Springer–Verlag, 1993.
14. D. Panario, X. Gourdon and Ph. Flajolet, *An analytic approach to smooth polynomials over finite fields*, Proceedings of the Third International Symposium on Algorithmic Number Theory, LNCS, vol. 1423, Springer-Verlag, pp. 226–236, 1998.
15. O. Schirokauer, *Discrete logarithms and local units*, Philos. Trans. Roy. Soc. London Ser. A, 345(1676), pp. 409–423, 1993.
16. O. Schirokauer, *Virtual logarithms*, J. Algorithms, vol. 57(2), pp. 140–147, 2005.
17. D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Transactions on Information Theory, IT-32, pp. 54–62, 1986.

A Implementation Details

As our algorithms are sub-exponential, the assessment of their *experimental* behavior is essential. We hence implemented the FFS and NFS attacks as described in sections 3.1 and 4.

A.1 Function Field Sieve Experiments

The FFS-based attack acquired the capability to compute the d-th power of an arbitrary target in $\mathbb{F}_{2^{1025}}$ after a learning phase of only 76,916,368 SDHP-oracle calls (these requests correspond to irreducible polynomials in α and β of degree up to 29).

We selected:

$$f_1(x) = x^{171} + x^4 + x^3 + x^2 + 1 \quad \text{and} \quad f_2(y) = y^6 + y + 1$$

so that an irreducible polynomial of degree 1025 divides the resultant $f(x)$ of $y - f_1(x)$ and $x - f_2(y)$ with respect to the variable y.

Let σ be the mapping sending an integer $\mathfrak{p}$ to a polynomial $\sigma(\mathfrak{p})$ over $\mathbb{F}_2$ such that the evaluation at 2 of this polynomial yields $\mathfrak{p}$. For instance, $\sigma(\mathtt{b}) = x^3 + x + 1$ where $\mathtt{b}$ is the hexadecimal integer `0x0b`.

Denoting by $\pi = 3.14159\ldots$, the attacked instance is:

$$\bar{\pi}(\alpha) = \sigma(\lfloor 2^{1023}\pi \rfloor)(\alpha) = \alpha^{1024} + \alpha^{1023} + \alpha^{1020} + \ldots + \alpha^5 + \alpha^4 + \alpha^2$$

As the factors of $x^{18}\bar{\pi}(x) \bmod f(x)$ have degrees 1, 4, 6, 11, 32, 73, 217, 339 and 341, listing here the entire descent tress would require too much space. Instead, we opt to zoom on the path of length 12 starting with a special-q equal to the factor of degree 341 and following recursively, at each node, the highest degree factor. This yields table 1, where the $a_{i,j}$ stand for the following hexadecimal values:

$$\begin{array}{cccc}
a_{1,0} = \texttt{1de914aa624143ee2880268} & a_{1,1} = \texttt{5368318d2e945f69775022f} & & \\
a_{1,2} = \texttt{6b625fb7825342aecdabd80} & a_{1,3} = \texttt{1055c12e550f64c6d8bd2e6} & & \\
a_{2,0} = \texttt{22ac2088c59} & a_{2,1} = \texttt{114043dab72} & a_{2,2} = \texttt{3e6c922af2e} & \\
a_{3,0} = \texttt{3f536e224dd} & a_{3,1} = \texttt{1077f087dba} & a_{4,0} = \texttt{000b1a0283e} & a_{4,1} = \texttt{002cce1067f} \\
a_{5,0} = \texttt{0006a24a2be} & a_{5,1} = \texttt{000164b63c6} & a_{6,0} = \texttt{0001bf55bcb} & a_{6,1} = \texttt{0000c949e26} \\
a_{7,0} = \texttt{00026dc2d0b} & a_{7,1} = \texttt{0000b748064} & a_{8,0} = \texttt{003e4818437} & a_{8,1} = \texttt{0001b3671a7} \\
a_{9,0} = \texttt{000138bc9c0} & a_{9,1} = \texttt{000122f0d95} & a_{10,0} = \texttt{000052ee9bb} & a_{10,1} = \texttt{0000e82d833} \\
a_{11,0} = \texttt{00003972dfb} & a_{11,1} = \texttt{00069fc51c5} & a_{12,0} = \texttt{0006faf0133} & a_{12,1} = \texttt{0000d86d785}
\end{array}$$

The special-q degree in x or y is underlined in the corresponding column and smoothness probabilities are expressed as $2^{-\psi}$.

We wrote our software chain in C, relying upon the computer algebra systems PARI-GP and MAGMA for a handful of specific tasks. The whole computation was run on single Intel Core-2 at 3.6 GHz, with a 16 GB memory. Each SqSS call claimed roughly five minutes and the entire attack took less than a week.

A.2 Number Field Side Experiments

We also experimented the NFS variant in the finite field $\mathbb{F}_p$, where p is the 516-bit prime $\lfloor 10^{155}\pi + 88896 \rfloor$ (p is the smallest prime above $10^{155}\pi$ such that $\frac{p-1}{2}$ is also prime).

We computed the secret d-th power of an element in $\mathbb{F}_Q$ after a learning phase of 140 million SDHP-oracle queries.

We selected the following polynomials for defining respectively the rational and algebraic sides:

$$\begin{aligned}
f_1 &= x - \texttt{0x5bd59a8c1dfa4580bbd2cee} \\
f_2 &= \texttt{0xf6841ca54cc1267c} \cdot x^5 + \texttt{0x423b1fc0c94938f26} \cdot x^4 \\
&\quad - \texttt{0x2495d40c188755399c7c} \cdot x^3 + \texttt{0x3b7ed50dd0329dda55051} \cdot x^2 \\
&\quad - \texttt{0x23e3eeb7641a7d13c4b182} \cdot x + \texttt{0x1a29246950783fbbb6b7b7c7}
\end{aligned}$$

Sieving was done using a modified version of J. Franke and T. Kleinjung's lattice sieving code as included in the ggnfs software suite. We selected a factor base comprising the 43,321 prime ideals of norm smaller than 2^{19} and obtained generators factoring over this set of ideals after four sieving minutes using 128 Intel Core-2 CPU cores clocked at 1.6 GHz.

$H(x,y)$ ↙ ↘		ψ
factor degrees of $H(x, f_1(x))$	factor degrees of $H(f_2(y), y)$	
$\sigma(a_{1,0}) + \sigma(a_{1,1})y + \sigma(a_{1,2})y^2 + \sigma(a_{1,3})y^3$		3
$3, 4, 29, 30, 46, 73, 75, \underline{341}$	$1, 5, 20, 20, 21, 47, 64, 68, 91, 100, 103$	
$\sigma(a_{2,0}) + \sigma(a_{2,1})y + \sigma(a_{2,2})y^2$		16
$1, 12, 20, 23, 25, 31, 33, 54, 59, 62, 63$	$1, 13, 14, 21, 43, 53, \underline{103}$	
$\sigma(a_{3,0}) + \sigma(a_{3,1})y$		18
$2, 12, 13, 15, 27, 33, 46, \underline{63}$	$1, 12, 15, 16, 17, 23, 27, 43, 43, 44$	
$\sigma(a_{4,0}) + \sigma(a_{4,1})y$		19
$2, 3, 12, 17, 18, 33, 35, 35, \underline{46}$	$2, 6, 10, 11, 12, 13, 29, 35, 38, 39$	
$\sigma(a_{5,0}) + \sigma(a_{5,1})y$		21
$1, 1, 2, 3, 5, 15, 18, 18, 22, 24, 24, 27, 35$	$1, 5, 6, 8, 14, 18, 28, 29, 32, \underline{39}$	
$\sigma(a_{6,0}) + \sigma(a_{6,1})\, y$		22
$5, 12, 12, 18, 19, 31, 32, 34, \underline{35}$	$1, 2, 4, 9, 14, 18, 19, 20, 23, 27, 27$	
$\sigma(a_{7,0}) + \sigma(a_{7,1})\, y$		24
$1, 5, 8, 10, 18, 26, 30, 33, 33, \underline{34}$	$1, 2, 5, 6, 6, 9, 10, 16, 17, 22, 23, 25, 30$	
$\sigma(a_{8,0}) + \sigma(a_{8,1})\, y$		26
$1, 1, 11, 20, 20, 24, 27, 29, 32, \underline{33}$	$1, 6, 7, 9, 17, 23, 24, 25, 25, 30, 30$	
$\sigma(a_{9,0}) + \sigma(a_{9,1})\, y$		27
$2, 9, 13, 18, 19, 20, 26, 27, 31, \underline{32}$	$18, 22, 24, 24, 26, 26, 29$	
$\sigma(a_{10,0}) + \sigma(a_{10,1})\, y$		28
$1, 1, 3, 14, 20, 20, 24, 27, 27, 28, \underline{31}$	$1, 4, 7, 16, 21, 25, 27, 28, 30$	
$\sigma(a_{11,0}) + \sigma(a_{11,1})\, y$		28
$1, 4, 10, 15, 15, 23, 24, 24, 26, 29, 30$	$1, 2, 4, 7, 8, 12, 12, 13, 18, 20, 24, 29, \underline{30}$	
$\sigma(a_{12,0}) + \sigma(a_{12,1})\, y$		30
$1, 1, 14, 16, 18, 18, 18, 25, 28, 29, \underline{30}$	$1, 2, 3, 10, 16, 18, 22, 23, 25, 27, 28$	

Table 1. An example path in an attack tree for $\mathbb{F}_{2^{1025}}$

We then used the factor base extension step detailed in [11] and sieved again for 24 minutes, using the same set of processors. This allowed us to relate the vast majority of ideals of norm smaller than 2^{32} to smaller-norm ideals using 140 million oracle queries.

We postponed linear algebra computations after the descent (*cf.* to the two options explained in section 4).

The descent was coded in MAGMA, where the SqSS was performed by a modified version of J. Franke and T. Kleinjung's lattice sieving code, including several modifications pertaining to the descent step (most notably co-factorization of norms using the `gmp-ecm` program).

We started from the challenge value $\tau = 2^{\lfloor \log_2 N \rfloor - \lfloor \log_3 N \rfloor}$. The first decomposition step consisted in finding a related finite field element (*e.g.* $2^k\tau$ for some k, assuming 2 was an oracle query) which factored suitably over the integers.

We searched during a few minutes for a suitable integer k. The corresponding factorization involved seven primes unlinked to the factor base, the largest of which having 27 digits. Afterwards, using 164 intermediate descent steps (139 on the algebraic side and 25 on the rational side), we managed to link our initial challenge to the factor base.

The final link between our challenge and the oracle queries was obtained by solving a linear system involving the matrix formed by the first set of relations obtained by sieving; the right hand side formed by the ideal factorization stemming from the descent. After adding character maps, we solved this 43,378 rows and columns system in eight hours on four Intel Core-2 CPU cores clocked at 2.4 GHz.

B Complexity Analysis – Regular NFS

Here, we work in $\mathbb{F}_Q$, with $Q = p^n$ and $p > L_Q(\frac{2}{3})$.

The polynomial construction is done through lattice reduction. First, choose an irreducible polynomial f_1 of degree n and find another polynomial of degree D, multiple of f_1 modulo p. The coefficients of f_2 have size $Q^{1/(D+1)}$, the coefficients of f_1 are essentially of the same size, just slightly larger.

The main parameter is $D = \delta \sqrt[3]{\frac{\log Q}{\log\log Q}}$. This implies that the coefficients of f_1 and f_2 have size $L_Q(\frac{2}{3}, \frac{1}{\delta})$.

B.1 Sieving And Linear Algebra

There is a small and a large side. The complexity is clearly dominated by the large side, corresponding to f_2. We sieve over elements $a + b\alpha$, map them into the number field and compute their norm $b^D f_2(-\frac{a}{b})$. If a and b are bounded in absolute value by $B = L_Q(\frac{1}{3}, \beta)$, the norm is $L_Q(\frac{2}{3}, \beta\delta + \frac{1}{\delta})$.

As usual, the smoothness probability over a smoothness basis of ideals of prime norm up to $L_Q(\frac{1}{3}, \beta)$ is $L_Q(\frac{1}{3}, -(\beta\delta + \frac{1}{\delta})\frac{1}{3\beta})$. To ensure that we get enough relations, we need:

$$\beta = -\left(\beta\delta + \frac{1}{\delta}\right)\frac{1}{3\beta} \quad \Rightarrow \quad 3\beta^2 - \beta\delta - \frac{1}{\delta} = 0$$

$$\text{which has the positive root: } \beta = \frac{\delta + \sqrt{\delta^2 + \frac{12}{\delta}}}{6}$$

This is minimized for $\delta = \sqrt[3]{\frac{3}{2}}$ *i.e.* $\beta = \sqrt[3]{\frac{4}{9}}$.

The complexity of the sieving step is thus $L_Q(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$.

B.2 Descent Initialization: The Descent's Dominating Step

In the initial descent step, we first need to go from norms of size $L_Q(1)$ down to factors of size $L_Q(\frac{2}{3})$. Indeed, by opposition to the FFS setting, here smoothness testing can only be done using ECM factorization. Consequently, this step costs $L_{\text{factor}}(\frac{1}{2}, \sqrt{2})$ of the bound on the factor size.

With factors of size $L_Q(\frac{2}{3})$, this becomes $L_Q(\frac{1}{3})$ and we cannot afford more than that. We use the same strategy as described in [11] and simply randomize the input challenge until it becomes smooth as an integer. As the complexity of pulling-out a factor F with ECM costs $L_F(\frac{1}{2}, \sqrt{2})$, we can analyze the descent from arbitrary numbers with norm $L_Q(1, 1)$ down to factors of size $L_Q(\frac{2}{3}, \theta)$.

Smoothness probability is $L_Q(\frac{1}{3}, \frac{\theta}{3})$ and ECM's cost is $L_Q(\frac{1}{3}, \sqrt{\frac{4\theta}{3}})$. Thus, the total work-factor per smooth value is $L_Q(\frac{1}{3}, \frac{\theta}{3} + \sqrt{\frac{4\theta}{3}})$.

This is minimized for $\theta^3 = \frac{1}{3}$ and costs $L_Q(\frac{1}{3}, \sqrt[3]{3})$.

B.3 Continuing The Descent

In the middle of the descent, we consider a special-q of size $L_Q(\lambda, \mu)$ with $\frac{1}{3} < \lambda < \frac{2}{3}$. We sieve over polynomials of degree:

$$T = t \left(\frac{\log Q}{\log \log Q} \right)^{\frac{\lambda - \frac{1}{3}}{2}}$$

and with coefficients of size $S = L_Q((\lambda + \frac{1}{3})\frac{1}{2}, s)$. The size of the sieving space is $L_Q(\lambda, st - \mu)$. Since $L_Q(\frac{1}{3})$ freedom should suffice, we can make st tends to μ from upward.

The f_1 norm is $L_Q(\frac{1}{2} + \frac{\lambda}{2}, \frac{t}{\delta})$ and the f_2 norm is $L_Q(\frac{1}{2} + \frac{\lambda}{2}, \frac{t}{\delta} + s\delta)$.

Thus the total norm after dividing by the special-q, which can be ignored is, $L_Q(\frac{1}{2} + \frac{\lambda}{2}, \frac{2t}{\delta} + s\delta)$. Minimizing this under the constraint $st = \mu$ yields $t = \delta\sqrt{\frac{\mu}{2}}$. This step is clearly not the dominating task in the descent. In fact, we can adapt the descent's speed to the specific problem at hand. The only restriction is that the descent should be fast enough to only get a relatively small number of values at the final step of the descent.

B.4 Concluding The Descent

In the final descent step, we need to express special-q values of size immediately above $L_Q(\frac{1}{3}, \beta)$ in the smoothness bases up to $L_Q(\frac{1}{3}, \beta)$. We sieve over polynomials $a + bx$ with a and b bounded by $L_Q(\frac{1}{3}, \frac{\beta+\epsilon}{2})$, with a total sieving space size equal to $L_Q(\frac{1}{3}, \epsilon)$.

At the f_2 side, the norm's size is computed as above and yields $L_Q(\frac{2}{3}, (\beta+\epsilon)\frac{\delta}{2} + \frac{1}{\delta})$.

At the f_1 side, only the constants matter and the norm has size $L_Q(\frac{2}{3}, \frac{1}{\delta})$. Thus the total norm is $L_Q(\frac{2}{3}, (\beta+\epsilon)\frac{\delta}{2} + \frac{2}{\delta})$ and is left unchanged when taking into account the special-q that can be removed. The smoothness probability in basis $L_Q(\frac{1}{3}, \beta)$ is:

$$L_Q\left(\frac{1}{3}, \frac{1}{3}\left(\frac{(\beta+\epsilon)\delta}{2\beta} + \frac{2}{\beta\delta}\right)\right)$$

This is constrained by the fact that we need enough sieving space, *i.e.* by the relation:

$$3\epsilon = \frac{(\beta+\epsilon)\delta}{2\beta} + \frac{2}{\beta\delta},$$

which yields $\epsilon \simeq 1.27$ as in [11], which does not dominate the overall asymptotic complexity of the descent step.

C Complexity Analysis – High Degree NFS

Here, we work in $\mathbb{F}_Q$, with $Q = p^n$ and $L_Q(\frac{2}{3}) > p > L_Q(\frac{1}{3})$. We write $p = L_Q(\lambda, 1)$.

The polynomial construction is as follows. First, choose a parameter $0 < \alpha < \frac{1}{2}$, which value will be determined later, on and an integer A close to p^α. Using Euclidean division write $p = BA + r_p$ with $B \simeq p^{1-\alpha}$ and $0 \leq r_p < A$.

Then, find an irreducible polynomial f_1 of degree n, which coefficients are either small numbers or a small multiples of A. All coefficients of f_1 have a size bounded by p^α. Finally, let $f_2 = Bf_1 \bmod p$, all coefficients of f_2 are either small multiples of B or small multiples of r_p. Hence, their size is at most of order $p^{1-\alpha}$.

C.1 Sieving And Linear Algebra

Again, complexity is bounded by the large side, given by f_2.

We sieve over polynomials of degree D defined by:

$$D = \delta\left(\frac{\log Q}{\log\log Q}\right)^{\frac{2}{3}-\lambda}$$

with coefficients of size $L_Q(\lambda - \frac{1}{3}, \mu)$. The total sieving space size is $L_Q(\frac{1}{3}, \delta\mu)$.

At the f_2 side, the norm is $L_Q(\frac{2}{3}, \delta(1-\alpha) + \mu)$. For a given sieving space size, this is minimized when $\delta(1-\alpha) = \mu$. With a smoothness basis bounded by $L_Q(\frac{1}{3}, \beta)$, the smoothness probability is $L_Q(\frac{1}{3}, -\frac{2\mu}{3\beta})$. To balance the complexities of the sieving and the linear algebra we have:

$$\beta = \frac{2\mu}{3\beta} \quad \textit{i.e.} \quad 3\beta^2 = 2\mu.$$

In addition, to ensure that we get enough relations, we need:

$$\beta + \frac{2\mu}{3\beta} = (1-\alpha)\mu^2 \quad \textit{i.e.} \quad 2\beta = (1-\alpha)\mu^2.$$

Put together, this yields $\beta = \sqrt[3]{\frac{8}{9}(1-\alpha)}$. And, the complexity of the sieving step is $L_Q(\frac{1}{3}, \sqrt[3]{\frac{64}{9}(1-\alpha)})$.

C.2 Descent Initialization

In the initial descent step, as in the regular NFS case, we first need to go from norms of size $L_Q(1)$ down to factors of size $L_Q(\frac{2}{3})$. However, since f_1's coefficients are of order p^α, the norm of a generic element, with degree n and coefficient modulo p is larger than Q. This norm is in fact, of order $L_Q(1, 1+\alpha)$.

Using the ECM to go to factors of size $L_Q(\frac{2}{3}, \theta)$ costs $L_Q(\frac{1}{3}, \frac{1+\alpha}{3\theta} + \sqrt{\frac{4\theta}{3}})$.

This is minimized for $\theta^3 = (1+\alpha)^{\frac{2}{3}}$ and costs $L_Q(\frac{1}{3}, \sqrt[3]{3(1+\alpha)})$.

Equilibrating the complexities of the sieving and of the descent's initialization, we get:

$$3(1+\alpha) = \frac{64(1-\alpha)}{9} \quad \textit{i.e.} \quad \alpha = \frac{37}{91}.$$

For this α the complexities of the sieving and of the descent's initialization become:

$$L_Q(\frac{1}{3}, \sqrt[3]{\frac{384}{91}}) \simeq L_Q(\frac{1}{3}, 1.62)$$

C.3 Continuing the descent

In the middle of the descent, we encounter a special-q of size $L_Q(\lambda', \mu')$ with $\frac{1}{3} < \lambda' < \frac{2}{3}$. We sieve over polynomials of degree:

$$T = t\left(\frac{\log Q}{\log\log Q}\right)^{\lambda_t}$$

and with coefficients of size $S = L_Q(\lambda_s, s)$. The two values λ_s and λ_t are related and sum to λ'. More precisely, we choose:

$$\lambda_t = \frac{\lambda'+1}{2} - \lambda$$
$$\lambda_s = \frac{\lambda'-1}{2} + \lambda$$

The sieving space size is $L_Q(\lambda', st-\mu')$. Since $L_Q(\frac{1}{3})$ freedom should suffice, we can make st tends to μ' from upward. Again, there is enough freedom here and this step is not limiting.

C.4 Concluding The Descent

In the final descent step, we need to express special-q values of size immediately above $L(\frac{1}{3}, \beta)$ in the smoothness bases up to $L(\frac{1}{3}, \beta)$. We sieve over polynomials of degree D defined by:

$$D = \delta \left(\frac{\log Q}{\log \log Q} \right)^{\frac{2}{3} - \lambda}$$

with coefficients of size $L_Q(\lambda - \frac{1}{3}, \mu)$. The total sieving space has size $L_Q(\frac{1}{3}, \delta\mu - \beta)$. Here the total norm does not depend on α and is $L_Q(\frac{2}{3}, \delta + 2\mu)$. Moreover, the norm's expression is left unchanged when removing the special-q contribution.

The smoothness probability in basis $L_Q(\frac{1}{3}, \beta)$ is

$$L_Q \left(\frac{1}{3}, \frac{\delta + 2\mu}{3\beta} \right)$$

Again, this is constrained by the fact that we need enough sieving space, *i.e.* by the relation:

$$\frac{\delta + 2\mu}{3\beta} = \delta\mu - \beta$$

Choosing $\delta = 2\mu$, we get:

$$2\mu^2 - \frac{4\mu}{3\beta} - \beta = 0.$$

which yields $\mu \simeq 1.13$. The total complexity $L_Q(\frac{1}{3}, 2\mu^2 - \beta) \simeq L_Q(\frac{1}{3}, 0.93)$ is, again, not dominating.