The Solution of McCurley's Discrete Log Challenge

Damian Weber[1], Thomas Denny[2]

[1] Institut für Techno– und Wirtschaftsmathematik
Erwin–Schrödinger–Str. 49
D–67663 Kaiserslautern
`weber@itwm.uni-kl.de`

[2] debis IT Security Services
Rabinstraße 8
D–53111 Bonn
`t-denny@itsec-debis.de`

Abstract

We provide the secret Diffie-Hellman-Key which is requested by Kevin McCurley's challenge of 1989. The DH-protocol in question has been carried out in $(\mathbb{Z}/p\mathbb{Z})^*$ where p is a 129-digit prime of special form. Our method employed the Number Field Sieve. The linear algebra computation was done by the Lanczos algorithm.

Keywords: Discrete Logarithms, Number Field Sieve, Index Calculus, Lanczos

1 Introduction

When discrete log cryptosystems are designed the groups $(\mathbb{Z}/p\mathbb{Z})^*$ serve as a standard choice, as for example in [3, 4, 14]. In view of the Diffie–Hellman key exchange protocol introduced in [3], McCurley stated a challenge by using the following setup [11]:

$$\begin{aligned} b_A &= 127402180119973946824269244334322849749382042586931621654557735290322914679095998681860978813046595166455458144280588076766033781 \\ b_B &= 180162285287453102444782834836799895015967046695346697313025121734059953772058475958176910625380692101651848662362137934026803049 \\ p &= (739 \cdot 7^{149} - 736)/3 \\ q &= (p-1)/(2 \cdot 739) \end{aligned}$$

The order of the multiplicative group, which is generated by the element 7, splits as follows: $|(\mathbb{Z}/p\mathbb{Z})^*| = 2 \cdot 739 \cdot q$.

- Alice computes (using her secret key x_A) as $7^{x_A} \equiv b_A \pmod{p}$
- Bob computes (using his secret key x_B) as $7^{x_B} \equiv b_B \pmod{p}$

Kevin McCurley asked for the common secret key $K \equiv 7^{(x_A \cdot x_B)} \pmod{p}$ which we computed at 03:50pm Middle European Time on Jan 25, 1998 as

$$\begin{aligned} K = {} & 38127280411190014138078391507929634193998643551018670285056375615 \\ & 045523966929403922102172514053270928872639426370063532797740808, \end{aligned} \tag{1}$$

by first calculating

$$\begin{aligned} x_A = {} & 6185869085965188327359333165203790426798764306952171345914622218 \\ & 49525998156144877820757492182909777408338791850457946749734, \end{aligned} \tag{2}$$

the secret key of Alice.

Since p is of a special form it is very convenient to use the number field sieve discrete log algorithm [17, 5] for the precomputation. This step has already been carried out in [21]. It was unclear whether one could keep that attractive number field chosen there for computing the individual logarithms as well [18]. Readers not familiar with the number field sieve are recommended to look up these references. We note that for "general" p, the record for discrete logarithms is 85 decimal digits [20].

In contrast to factoring with the number field sieve, some additional computational difficulties have to be dealt with. For example, it is costly to transform the original congruence to another form such that the number field sieve is actually applicable. Another constraint is the smaller size of factor bases – otherwise the linear algebra step would be infeasible.

After the introduction of notation in section 2 we start with a description of how we transformed the original problem to a problem where only logarithms of "small" elements have to be computed in section 3. Section 4 deals with the choice of the polynomial defining the number field. Section 5 is devoted to lattice sieving techniques for discrete log computations. In section 6 we pay attention to computing the solution of the resulting sparse linear system mod q, which eventually yielded the solution to the original discrete log problem.

2 Notation of the Number Field Sieve Parameters

Let the discrete logarithm problem be $a^x \equiv b \pmod{p}$, p prime. Let f be an irreducible polynomial of degree n, the coefficient of X^n be called h. Furthermore, let $\alpha \in \mathbb{C}$ be a root of f. There should be at least one root m of $f(X)$ modulo p. Set $\omega = h \cdot \alpha$. In this case there exists a ring homomorphism

$$\begin{aligned} \varphi : \mathbb{Z}[\omega] & \longrightarrow \mathbb{Z}/p\mathbb{Z} \\ \omega & \mapsto h \cdot m. \end{aligned}$$

We set $g(X) := X - m$. We denote with $\mathcal{FB}_{\text{alg}}$ (the algebraic factor base) a finite set of prime ideals in the ring of integers of $\mathbb{Q}(\omega)$, and with $\mathcal{FB}_{\text{rat}}$ (the rational factor base) a finite set of prime numbers in $\mathbb{Z}$.

3 Reduction of the Original Problem

With the number field sieving step, logarithms of elements $s \in \mathbb{Z}/p\mathbb{Z}$ can be obtained, if a smooth $\sigma \in \varphi^{-1}(s) \subset \mathbb{Z}[\omega]$ is known. In particular, this is the case if s lies in the rational factor base. Having planned to use a rational factor base size of 30000 this would mean to aim at bounding s by the 30000–th prime, that is 350381. With the current available methods this is not feasible. With the following relation, however, we are reduced to s's which are not bigger than $5.05 \cdot 10^{13}$:

$$a^{141266132} \cdot b_A \equiv \frac{t}{v} \pmod{p},$$

where

$$\begin{aligned} t &= 2^3 \cdot 31 \cdot s_1 \cdot s_3 \cdot s_6 \cdot s_8 \cdot s_{10} \cdot s_{11} \\ v &= 353 \cdot s_2 \cdot s_4 \cdot s_5 \cdot s_7 \cdot s_9 \cdot s_{12}. \end{aligned} \tag{3}$$

with

$$\begin{aligned} s_1 &= 603623, \\ s_2 &= 165073039, \\ s_3 &= 1571562367, \\ s_4 &= 1601141623, \\ s_5 &= 1715568391, \\ s_6 &= 7575446399, \\ s_7 &= 13166825869, \\ s_8 &= 265542836371, \\ s_9 &= 371303006453, \\ s_{10} &= 4145488613977, \\ s_{11} &= 4338202139093, \\ s_{12} &= 5041332876473. \end{aligned}$$

We proceed with how equation (3) has been found. Each $b \in (\mathbb{Z}/p\mathbb{Z})^*$ can be expressed as a quotient $b \equiv t/v \pmod{p}$ with $|t|, |v| < \sqrt{p}+1$ [19, Th.67]. Such a representation can be found by applying the extended Euclidean algorithm. In the challenge computation we computed $b_l :\equiv 7^l \cdot b \pmod{p}$ for many l's, found the quotient $b_l \equiv t_l/v_l$ and tried to split t_l and v_l. The ρ–function [6] tells us how many pairs we need to test until a successful decomposition occurs. The term $\rho_i(\alpha)$ denotes the probability that the i–th largest prime factor of the number n is at most $n^{1/\alpha}$. We set $\rho(\alpha) := \rho_1(\alpha)$.

We started looking for factors of t and v with at most 15 decimal digits simultaneously. The probability that a 65–digit number has at most 15 digits is $\rho(13/3) \approx 2.12 \cdot 10^{-3}$. Assuming t and v behave as random integers of this size we expect to find a successful l after $\approx 1/(2.12 \cdot 10^{-3})^2 \approx 222500$ trials. We distributed the interval [141200001; 141422500] among 40 Sparc ELC and 20 Sparc 4. As a Sparc 4 is about four times faster than a Sparc ELC, we chose the interval length for the ELC as 1850 in contrast to 7500 for the Sparc 4 stations.

Each machine carried out four different stages; we give the average running times for each stage (Sparc 4 workstation) per l:

1. trial division up to 10^6 (1.35 sec),
2. ECM for factors up to 10^9 (7.11 sec),
3. ECM for factors up to 10^{12} (30.02 sec),
4. ECM for factors up to 10^{15} (128.80 sec),

where ECM is an acronym for Lenstra's elliptic curve factoring method [10]. We label these stages by TDIV, ECM9, ECM12 and ECM15, respectively.

A pair (t, v) is *useless* if either t or v contains a prime factor above our smoothness bound 10^{15}. We can estimate beforehand how many pairs (t, v) will be recognized as useless by each stage if we evaluate the ρ_2 function.

Table 1. Probability of a number of 65 digits recognized as useless.

useless after stage	TDIV	ECM9	ECM12	ECM15
with probability	0.184	0.238	0.243	0.190

Using trial division in combination with ECM is a very tedious smoothness test; we experimented with an early abort strategy. Such a technique has been used in several algorithms, originating from [16]. In our case, after each stage we removed a third of the pairs (t, v) – those with the biggest unfactored part – from our list. The following behaviour is to be expected when applying this strategy on each Sparc 4 processor. On such a machine we start with 7500 pairs (t, v). From table 1 we estimate that approximately $(1-0.184)^2 \approx 66.6$ %, that is 4994, will survive the trial division step. Our early abort condition removes a third of the pairs, this leaves 3329. From these approximately $(1 - 0.238)^2 \approx 58.2$ % survive ECM9 (1933). Another cut by early aborting leaves 1288. ECM12 yields 738 pairs left. After removing a third of these pairs, 492 are left for ECM15. In case of no successful pair being found we can now estimate the time of the smoothness tests per processor as:

$$7500 \cdot 1.35 + 3329 \cdot 7.11 + 1288 \cdot 30.02 + 492 \cdot 128.8 = 135829.55 \text{ sec} \approx 37.7 \text{ h}.$$

Table 2. Reduction sieve: # pairs per stage.

proc	TDIV		ECM9		ECM12		ECM15		run–time (h)
	start	useless	start	useless	start	useless	start	useless	total
1	7500	2535	3307	1139	1444	574	579	364	41.0
2	7500	2446	3366	1192	1448	603	563	339	41.3
3	7500	2531	3309	1232	1383	565	545	309	41.5
4	7500	2408	3391	1181	1472	592	586	359	42.7
5	7500	2636	3239	1102	1423	590	555	341	40.3
6	7500	2497	3332	1190	1426	548	585	352	41.5

From table 2, which summarizes the output of six Sparc 4 processors, we see that the theoretical prediction of the useless pairs can be used to get a good estimate of the actual total running time per processor.

4 Choice of the Polynomial

This section is devoted to finding an appropriate polynomial f for the number field sieve (see section 2). According to practical experience we should consider number fields of degree $n = 3, 4, 5, 6$. In order to make the best choice, i.e. to find a maximum number of relations, we examine the probability of finding relations over two factor bases of optimal size with respect to the expected norm of elements $c+d\alpha$ of that number ring. In order to construct suitable polynomials of such degrees we may use the identities

$$\begin{aligned} 21p &= 739 \cdot (7^{50})^3 - 5152 \\ 3p &= 5173 \cdot (7^{37})^4 - 736 \\ 21p &= 739 \cdot (7^{30})^5 - 5152 \\ 21p &= 739 \cdot (7^{25})^6 - 5152. \end{aligned}$$

We therefore have to choose among the following pairs of polynomials

$g(X) = X - 7^{50}$	$f(X) = 739 \cdot X^3 - 5152$
$g(X) = X - 7^{37}$	$f(X) = 5173 \cdot X^4 - 736$
$g(X) = X - 7^{30}$	$f(X) = 739 \cdot X^5 - 5152$
$g(X) = X - 7^{25}$	$f(X) = 739 \cdot X^6 - 5152$

In order to compare the four different possible choices of the degree, we look at the values $c + dm$, $N(c + d\alpha)$ to be decomposed over the factor bases. On the rational side we get $c + dm \approx dm$; on the algebraic side, we obtain $-739 \cdot c^n - 5152d^n \approx -739 \cdot c^n$. As a sieving rectangle of $10^6 \times 10^6$ was expected to be needed, we got table 3 with the aid of the ρ–function.

Table 3. Comparing different degrees

degree	m	dm	$h \cdot c^n$	$\lvert FB_1\rvert$	$\lvert FB_2\rvert$	# trials per full
3	$1.8 \cdot 10^{42}$	$1.8 \cdot 10^{48}$	10^{21}	19800	20200	$3.7 \cdot 10^{11}$
4	$1.9 \cdot 10^{31}$	$1.9 \cdot 10^{37}$	10^{28}	19900	20100	$6.2 \cdot 10^{9}$
5	$2.3 \cdot 10^{25}$	$2.3 \cdot 10^{31}$	10^{33}	19600	20400	$3.7 \cdot 10^{9}$
6	$1.3 \cdot 10^{21}$	$1.3 \cdot 10^{27}$	10^{39}	16400	23600	$1.1 \cdot 10^{10}$

We see from this table that degree 4 and 5 were competing with slight advantage of degree 5. The only way to find out which is the better one is to give both a try and continue with the one which produced more relations. Test sieving has been carried out for $x+ym$ and $N(x+y\alpha)$ in the rectangle $[-10^6, 10^6] \times [1, 5000]$.

From the following amount of relations it is evident that degree 5 is the better choice indeed (running time in sec on a Sparc 4).

Table 4. Relations after test sieving.

type	degree 4	degree 5
full	16	127
1 LP (alg)	49	411
1 LP (rat)	114	631
2 LP	344	2056
running time		
	5653	9535

5 Sieving

After fixing the polynomials f, g we had to choose appropriate sieving parameters. The precomputation in 1995 was carried out with the quadruple large prime variant. We repeated the precomputations for two reasons: firstly, several improvements in the linear algebra step allowed to use a bigger factor base, secondly, we wanted to find out how effective the double large prime variant would perform for this setting. Table 5 depicts the parameters and the sieving results of 1995 (quadruple large prime variation) and 1997 (double large prime variation).

Both setups produced the result we wished: a linear system the solution of which yields the logarithms of particular elements of $\mathbb{Z}/p\mathbb{Z}$. We comment on the figures in table 5. We raised the large prime bound in order to increase the likelihood to encounter a partial relation, which now contained at most one large prime for f and g. We shrinked the sieving interval for two reasons. In the polynomial $N(X + Y\alpha) = -739X^5 - 5152Y^5$ both variables contribute to the same extent to the absolute value of the norm. In our first run, however, we observed that after covering the first half of the square we have already had enough relations. So we attempted to end up with a square sieving range instead of a rectangle which was roughly achieved. The running time of the first run is a very crude estimate on behalf of mips years (mega instructions per second). Although this measurement is outdated, we want to stick to it, in order to have the running times comparable to many of the previous publications concerning discrete log and factoring computations. Our run–time measurement is based on the observation that the UltraSparc we used was eight times faster than a Sparc ELC workstation which is rated at 21 mips.

Table 5. Parameters and sieving results

	quadruple large prime	double large prime
parameters		
FB rat	20000	30000
FB alg	20000	30000
LP rat	10^6	$5 \cdot 10^7$
LP alg	$5 \cdot 10^5$	$5 \cdot 10^7$
x–range	$15 \cdot 10^6$	$5 \cdot 10^6$
y–range	$2 \cdot 10^6$	$4 \cdot 10^6$
run–time	110 mips y	$\approx$ 180 mips y
relations		
full	3199	10797
1 LP	42407	196734
2 LP	211888	895415
3 LP	478543	–
4 LP	388685	–
combined partials		
full	306717	75592

By utilizing the amount of relations depicted in table 5, logarithms of specific factor base elements can be computed. Now let's turn to the computation of logarithms of elements not lying in the factor base.

One of the open questions in [18] was: is it possible to compute the logarithm of an arbitrary element of $(\mathbb{Z}/p\mathbb{Z})^*$ without abandoning the especially comfortable polynomial f? Changing f is required by the theory but its analysis is for $p \to \infty$. In our computation, however, we pursued another route which has already been indicated in section 3. This is the attempt to transform the original problem such that the remaining elements, the logarithm of which is unknown, can be treated like factor base elements.

Let s be one of the primes of the right hand side of (3). A relation of the following form is required for each s:

$$\begin{aligned} c + dm &= s \prod_r r^{e_r}, \quad \text{product over } r \in \mathcal{FB}_{\text{rat}}, \\ [c + d\alpha] &= \prod_{\mathfrak{r}} \mathfrak{r}^{e_{\mathfrak{r}}}, \quad \text{product over } \mathfrak{r} \in \mathcal{FB}_{\text{alg}}. \end{aligned} \tag{4}$$

The lattice sieve satisfies this request by defining

$$M_s := \{(c, d) \mid c + dm \equiv 0 \pmod{s}\}$$

and searching for smooth elements among $(c + dm)/s$ and $c + d\alpha$ with (c, d) taken from a subset of M_s. This will be called *the lattice sieve for s*. Analogously, one may introduce lattice sieving for prime ideals. This is a standard technique in number field sieve implementations introduced by [15].

In general it is difficult to find directly a relation like (4). What may be expected is to find relations of the form

$$\begin{aligned} c + dm &= s \cdot R_1 R_2 \prod_r r^{e_r}, \quad \text{product over } r \in \mathcal{FB}_{\text{rat}}, \\ [c + d\alpha] &= \mathfrak{R}_1 \mathfrak{R}_2 \prod_{\mathfrak{r}} \mathfrak{r}^{e_{\mathfrak{r}}}, \quad \text{product over } \mathfrak{r} \in \mathcal{FB}_{\text{alg}}, \end{aligned} \tag{5}$$

with large prime (ideals) $R_1, R_2, \mathfrak{R}_1, \mathfrak{R}_2$. As with the large prime variation of the classical number field sieve, with additional relations containing each one of the $R_1, R_2, \mathfrak{R}_1, \mathfrak{R}_2$, these can be turned into a relation only containing *one* large prime, namely s.

The heuristic algorithm which eventually led to the solution of the challenge was as follows:

1. by lattice sieving for s find a quadruple large prime relation of the form (5)
2. let $\mathfrak{R}$ run through the list $R_1, R_2, \mathfrak{R}_1, \mathfrak{R}_2$; with the lattice sieve for $\mathfrak{R}$ find either
   - a single large prime relation containing only the large prime $\mathfrak{R}$ and proceed with the next $\mathfrak{R}$ from the list, or
   - a double large prime relation containing only the large prime $\mathfrak{R}$ and another large prime which we call $\mathfrak{R}'$; in this case repeat the second step of this algorithm with $\mathfrak{R}$ replaced by $\mathfrak{R}'$ (called iteration in table 7).

Step 1 from above (finding (5) for $s = s_i$, $1 \le i \le 12$) has been performed with the following sieving range:

Table 6. Lattice sieve step 1, sieving rectangle.

primes	x–range	y–range
s_1 to s_5	[-35000;35000]	[1;12000]
s_6	[-55000;55000]	[1;25000]
s_7 to s_{11}	[-30000;30000]	[1;5000]
s_{12}	[-50000;50000]	[1;50000]

After performing step one – the time of which is shown in table 7 – the lattice sieve iterations had to be carried out for primes which lay in the interval $[10^5; 10^{10}]$, the lower bound due to the maximal factor base element having a norm of 349949 (algebraic) and 350377 (rational). The upper bound is due to the difficulty of finding appropriate relations for the s's above s_5; s_6 has already 10 decimal digits.

Each lattice sieve iteration for the $\mathfrak{R}'$ was carried out with a sieving rectangle of $[-35000, 35000] \times [1, 20000]$ spending about 5000 sec for each on a Sparc 20. Summing this up, about 211 h in total on one Sparc 20 were needed. Clearly, it is trivial to distribute each s and each iteration to different workstations.

This concludes the description of all sieving tasks.

Table 7. Running times of lattice sieve steps.

prime	time (sec)	# LP in (5)	# iterations of step 2
s_1	5680	1	1
s_2	3089	2	4
s_3	3691	3	8
s_4	4696	2	7
s_5	5003	1	5
s_6	50941	4	13
s_7	1300	2	9
s_8	1314	2	16
s_9	1389	2	11
s_{10}	1343	4	21
s_{11}	1394	4	13
s_{12}	96810	4	8

6 Linear Algebra

In this section we describe the method of how to solve the linear system which consists of the exponents of the free and full relations, the combined partials (table 5) and the special relations produced by the heuristic algorithm in section 5. There are five more columns containing additive characters to ensure that q–th powers in $\mathbb{Q}(\alpha)$ can be constructed [17]. Hence, we are left with a matrix of a form shown in figure 1.

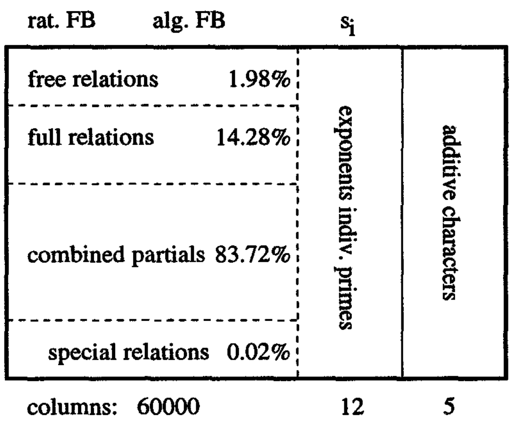

Fig. 1. Relation matrix.

The subsequent computation has been divided into two steps.

1. a preprocessing step (refinement of structured Gaussian elimination),
2. the Lanczos algorithm (description in [7], [8]).

The practicability of a possible alternative method, a combination of structured Gauss and ordinary Gaussian elimination suffers from enourmous space requirements (in our example about 2 GB of main memory).

We define n to be the number of unknowns and ω to be the total number of non–zero entries in the linear system. The running time of the Lanczos algorithm is known to be $O(n^2 + n\omega)$. The goal of step 1 is to iteratively decrease n while increasing ω as long as this running time is decreasing. To make a firm decision at this point, we need to predict the *actual* running time of step 2 on the machine we are using. Starting from the basic operations in the Lanczos algorithm, the following sections develop the model which we apply for that purpose.

6.1 Operations

The basic operations over $\mathbb{Z}/q\mathbb{Z}$ that are performed during an iteration of the Lanczos algorithm are

– computation of inner products
– matrix–vector multiplication
– vector updates (adding a multiple of a vector to another vector).

In order to speed up the computations over $\mathbb{Z}/q\mathbb{Z}$ we used the Montgomery representation [12]. Due to the fact that the linear system consists of exponents from decomposing integers, almost 95 % of the non-zero entries are equal to ± 1. The remaining entries c_i are relatively small ($-40 \leq c_i \leq 40$). Table 8 classifies the non-zero entries of our linear system before and after step 1.

Table 8. Compactification of relation matrix.

	original system	after preprocessing
unknowns	60 001	35 666
equations	75 592	35 688
avg. weight/equation	49.8	164.6
1–entries	1 785 588	2 657 470
(−1)–entries	1 761 865	2 711 635
c_i–entries	219 483	332 644

We could greatly reduce the time to perform a matrix–vector multiplication in $\mathbb{Z}/q\mathbb{Z}$ by computing all intermediate results in $\mathbb{Z}$ (but in Montgomery representation) and doing the reduction mod q only once, while creating the result vector. By this technique (lazy reduction) we achieve the substantial gain of 29% of the running time for the matrix-vector multiplication (timing on Sparc 20, different linear system):

Table 9. Average running time of one matrix–vector multiplication.

running time	original version	lazy reduction
addition	14.05 s	10.47 s
subtraction	13.22 s	11.08 s
scalar mult	6.17 s	1.70 s
final reduction	–	0.58 s
total	33.44 s	23.83 s

6.2 Running time model

An examination of all operations performed during the Lanczos algorithm to solve a linear system of dimension n with ω_1 1-entries, ω_2 (-1)-entries and ω_3 c_i-entries lead to the following formula:

$$\mathcal{T}(n,\omega,r) = n^2 \cdot t_2 + n \cdot (2 \cdot \omega + t_1) \tag{6}$$

where

$$\begin{aligned}
\omega &= \mathit{cache}_1 \cdot (\omega_1 \cdot T(\mathit{Add}) + \omega_2 \cdot T(\mathit{Sub})) + \mathit{cache}_3 \cdot \omega_3 \cdot T(\mathit{kmult}) \\
t_1 &= \mathit{cache}_2 \cdot (T(\mathit{Inv}) + (2+r) \cdot T(\mathit{Mult})) \\
t_2 &= \mathit{cache}_2 \cdot (T(\mathit{Square}) + 2 \cdot T(\mathit{Sub\_m}) + (2+r) \cdot T(\mathit{Add\_m}) \\
&\quad +(3+r) \cdot T(\mathit{Mult})) + 2 \cdot \mathit{cache}_3 \cdot T(\mathit{Red}).
\end{aligned}$$

In this formula r is the number of solutions that need to be calculated and $T(\mathit{operation})$ is the time needed to perform a single arithmetic operation on integers of magnitude of q. The variables $\mathit{cache}_1, \mathit{cache}_2$ and cache_3 represent the time needed to access main memory, first level cache and second level cache, respectively. For a Sparc 20 workstation we may take $\mathit{cache}_1 = 2.0$, $\mathit{cache}_2 = 1.0$, $\mathit{cache}_3 = 1.3$. These values are strongly machine dependent and have to be determined by experiment. By using (6), the time needed to solve a linear system of equations can be accurately predicted. This is crucial to decide whether an iteration of step 1 from above does improve the running time.

We now proceed by deriving a bound Δ (> 0) for the maximal increase of ω while decrementing n. This depends on the number and type of entries in our linear system as well as on the time of arithmetic and memory operations. To achieve a speed up and save memory, we have the condition

$$\mathcal{T}(n,\omega,r) \; - \; \mathcal{T}(n-1,\omega+\Delta,r) \; \geq \; 0.$$

Solving this for Δ, we obtain

$$\Delta \leq \frac{(2 \cdot n - 1) \cdot t_2 + 2 \cdot \omega + t_1}{2 \cdot (n - 1)}.$$

In practice, this upper bound lies between 100 and 1200.

6.3 Results

Using this formula and the ideas of the structured Gaussian elimination the running time of the Lanczos algorithm to compute 22 solutions of the original system was reduced by more than 50 %. The reduction in the dimension also led to an enormous reduction in the main memory requirements (approx. 30 MB) as 25 vectors have to be stored (see table 8 above).

Table 10 depicts the running time for the basic operations and the total running time of the original and the compactified system of linear equations (timings on Sparc 20).

Table 10. Speed up by compactification.

operation	original system	after preprocessing
matrix–vector multiplication	24.1 s	34.9 s
update vector	7.4 s	4.4 s
update solutions	85.0 s	48.3 s
inner product computation	3.7 s	2.2 s
single iteration	120.1 s	92.0 s
complete computation	2002.3 h	911.0 h

The solutions computed by the Lanczos algorithm finally yielded the logarithms of 2, 353, $s_1, \ldots, s_{12}$. The logarithm of 31 has already been known from [21]. By using the identities of (3), Alice's key was easy to obtain (2). The final task was to compute the common key K of Bob and Alice from $K \equiv b_B^{x_A} \pmod{p}$ shown in (1).

Acknowledgements

The authors are particularily grateful to Kevin McCurley for offering that challenge.

For providing the computing power we thank Ulrich Gräf (Sun Microsystems Benchmark Center/Germany) and Raimund Seidel (University of Saarland/Germany). The integer computations have been performed by two reliable and efficient multi–precision libraries, LiDIA and FREELIP. Accordingly, many thanks go to the LiDIA Group (University of Darmstadt/Germany) and Arjen Lenstra (Citibank/USA). Additionally, we wish to thank Johannes Buchmann, Oliver Schirokauer, Thomas Setz and Jörg Zayer.

References

1. I. Biehl and J. Buchmann and Th. Papanikolaou. LiDIA – a library for computational number theory. Technical report, Universität des Saarlandes/Germany, 1995. `http://www.informatik.th-darmstadt.de/TI/LiDIA`
2. Th. F. Denny. *Lösen grosser dünnbesetzter Gleichungssysteme über endlichen Primkörpern.* PhD thesis, Universität des Saarlandes/Germany, 1997.
3. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Information Theory 22*, pages pp. 472–492, 1976.
4. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31:469–472, 1985.
5. D. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.*, 6:124–138, 1993.
6. D. E. Knuth and L. Trabb Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3:321–348, 1976.
7. M. LaMacchia and A. Odlyzko. Solving large sparse linear systems over finite fields. In *Advances in Cryptology – Crypto '90*, number 537 in Lecture Notes in Computer Science, pages 109–133, 1990.
8. M. LaMacchia and A. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography*, 1:46–62, 1991.
9. A. K. Lenstra, H. W. Lenstra, Jr. (eds.). *The development of the number field sieve.* Number 1554 in Lecture Notes in Mathematics. Springer, 1993.
10. H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649–673, 1987.
11. K. S. McCurley. The discrete logarithm problem. In *Cryptology and Computational Number Theory*, number 42 in Proc. Symp. in Applied Mathematics, pages 49–74. American Mathematical Society, 1990.
12. P. L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44:519–521, 1985.
13. V. Müller and Th. F. Denny. On the reduction of composed relations from the number field sieve. In H. Cohen, editor, *Algorithmic Number Theory – ANTS II*, number 1122 in Lecture Notes in Computer Science, 1996.
14. National Bureau of Standards. *Digital signature standard*, 1994. FIPS Publication 186.
15. J. M. Pollard. *The lattice sieve.* Number 1554 in Lecture Notes in Mathematics. Springer, 1993.
16. C. Pomerance and S. S. Wagstaff. Implementation of the continued fraction integer factoring algorithm. In *Proc. 12th Manitoba Conf., Winnipeg/Manitoba 1982, Congr. Numerantium*, volume 37 of *Numerical mathematics and computing*, pages 99–118, 1983.
17. O. Schirokauer. Discrete logarithms and local units. *Phil. Trans. R. Soc. Lond. A 345*, pages 409–423, 1993.
18. O. Schirokauer, D. Weber, and Th. F. Denny. Discrete logarithms: the effectiveness of the index calculus method. In H. Cohen, editor, *Algorithmic Number Theory – ANTS II*, number 1122 in Lecture Notes in Computer Science, 1996.

19. D. Shanks. *Solved and unsolved problems in number theory (3rd ed.)*. Chelsea Publishing Company, 1985.
20. D. Weber. Computing discrete logarithms with quadratic number rings. In *Eurocrypt'98*, Lecture Notes in Computer Science, 1998. To appear.
21. D. Weber. Computing discrete logarithms with the number field sieve. In H. Cohen, editor, *Algorithmic Number Theory - ANTS II*, number 1122 in Lecture Notes in Computer Science, 1996.
22. D. Weber. *On the computation of discrete logarithms in finite prime fields*. PhD thesis, Universität des Saarlandes/Germany, 1997.
23. D. Weber. An implementation of the number field sieve to compute discrete logarithms mod p. *Advances in Cryptology - Eurocrypt'95*. number 921 in Lecture Notes in Computer Science, 1995.
24. J. Zayer. *Faktorisieren mit dem Number Field Sieve*. PhD thesis, Universität des Saarlandes/Germany, 1995.