Mersenne Factorization Factory

Thorsten Kleinjung[1], Joppe W. Bos[2], and Arjen K. Lenstra[1]

[1] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
[2] NXP Semiconductors, Leuven, Belgium**

Abstract. We present new factors of seventeen Mersenne numbers, obtained using a variant of the special number field sieve where sieving on the algebraic side is shared among the numbers. It reduced the overall factoring effort by more than 50%. As far as we know this is the first practical application of Coppersmith's "factorization factory" idea. Most factorizations used a new double-product approach that led to additional savings in the matrix step.
Keywords: Mersenne numbers, factorization factory, special number field sieve, block Wiedemann algorithm
Version: 17.1, January 12, 2015

1 Introduction

Despite its allegedly waning cryptanalytic importance, integer factorization is still an interesting subject and it remains relevant to test the practical value of promising approaches that have not been tried before. An example of the latter is Coppersmith's by now classical suggestion to amortize the cost of a precomputation over many factorizations [8]. The reason for the lack of practical validation of this method is obvious: achieving even a single "interesting" (i.e., record) factorization usually requires such an enormous effort [20] that an attempt to use Coppersmith's idea to obtain multiple interesting factorizations simultaneously would be prohibitively expensive, and meeting its storage requirements would be challenging.

But these arguments apply only to general numbers, such as RSA moduli [31], the context of Coppersmith's method. Given long-term projects such as [10, 11, 6] where many factoring-enthusiasts worldwide constantly busy themselves to factor many special numbers, such as for instance small-radix repunits, it makes sense to investigate whether factoring efforts that are eagerly pursued no matter what can be combined to save on the overall amount of work. This is what we set out to do here: we applied Coppersmith's factorization factory approach in order to simultaneously factor seventeen radix-2 repunits, so-called Mersenne numbers. Except for their appeal to makers of mathematical tables, such factorizations may be useful as well [18].

Let $S = \{1007, 1009, 1081, 1093, 1109, 1111, 1117, 1123, 1129, 1147, 1151, 1153, 1159, 1171, 1177, 1193, 1199\}$. For all $n \in S$ we have determined the full factorization of $2^n - 1$, using the method proposed in [8, Section 4] adapted to the special number field sieve (SNFS, [23]). Furthermore, for two of the numbers a (new, but rather obvious) multi-SNFS approach was exploited as well.

Most of our new factorizations soundly beat the previous two SNFS records, the full factorizations of $2^{1039}-1$ and $2^{1061}-1$ reported in [1] and [7] respectively. Measuring individual (S)NFS-efforts, factoring $2^{1193}-1$ would require about 20 times the effort of factoring $2^{1039}-1$ or more than twice the effort of factoring the 768-bit RSA modulus from [20]. Summing the

** Part of this work was conducted while this author was at Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA.

individual efforts for the seventeen numbers involved would amount to more than one hundred times the $(2^{1039} - 1)$-effort. Extrapolating our results, sharing the work à la Coppersmith allowed us to do it in about 50 times that effort. The practical implications of Coppersmith's method for general composites remain to be seen.

Although the factoring efforts reported here shared parts of the sieving tasks, each factorization still required its own separate matrix step. With seventeen numbers to be factored, and thus seventeen matrices to be dealt with, this gave us ample opportunity to experiment with a number of new algorithmic tricks in our block Wiedemann implementation, following up on the work reported in [1] and [20]. While the savings we obtained are relatively modest, given the overall matrix effort involved, they are substantial in absolute terms. Several of the matrices that we have dealt with are considerably larger than the one from [20], the largest published comparable matrix problem before this work.

Section 2 gives background on the (S)NFS and Coppersmith's method as required for the paper. Section 3 introduces our two sets of target numbers to be factored, while sections 4 and 5 describe how the two main steps of the SNFS here applied to these numbers. The newly found factors are presented in Section 6 and Section 7 concludes the paper with a few remarks.

All core years reported below are normalized to 2.2 GHz cores.

2 Background on (S)NFS and Coppersmith's method

2.1 Number field sieve

To factor a composite integer N in the current range of interest using the number field sieve (NFS, [23]), a linear polynomial $g \in \mathbf{Z}[X]$ and a degree $d > 1$ polynomial $f \in \mathbf{Z}[X]$ are determined such that g and f have, modulo N, a root $m \approx N^{1/(d+1)}$ in common. For any m one may select $g(X) = X - m$ and $f(X) = \sum_{i=0}^{d} f_i X^i$ where $N = \sum_{i=0}^{d} f_i m^i$ and $0 \le f_i < m$ (or $|f_i| \le \frac{m}{2}$) for $0 \le i \le d$. Traditionally, everything related to the linear polynomial g is referred to as "rational" and everything related to the non-linear polynomial f as "algebraic".

Relations are pairs of coprime integers a, b with $b \ge 0$ such that $bg(a/b)$ and $b^d f(a/b)$ have only small factors, i.e., are *smooth*. Each relation corresponds to the vector consisting of the exponents of the small factors (omitting details that are not relevant for the present description). Therefore, as soon as more relations have been collected than there are small factors, the vectors are linearly dependent and a matrix step can be used to determine an even sum of the vectors: each of those has probability at least 50% to lead to a non-trivial factor of N.

Balancing the smoothness probability and the number of relations required (which both grow with the number of small factors) the overall heuristic expected NFS factoring time is $L((64/9)^{1/3}) \approx L(1.923)$ asymptotically for $N \to \infty$, where

$$L(c) = L[\frac{1}{3}, c] \text{ and } L[\rho, c] = \exp((c + o(1))(\log(N))^{\rho}(\log(\log(N)))^{1-\rho})$$

for $0 \le \rho \le 1$ and the degree d is chosen as an integer close to $(\frac{3\log(N)}{\log(\log(N))})^{1/3}$. A more careful selection of g and f than that suggested above (following for instance [19]) can lead to a substantial overall speed-up but has no effect on the asymptotic runtime expression.

For regular composites the f_i grow as $N^{1/(d+1)}$ which is only $N^{o(1)}$ for $N \to \infty$ but in general not $O(1)$. Composites for which the f_i are $O(1)$ are "special" and the SNFS applies: its heuristic expected runtime is $L((32/9)^{1/3}) \approx L(1.526)$ asymptotically for $N \to \infty$, where the degree d is chosen as an integer close to $(\frac{3\log(N)}{2\log(\log(N))})^{1/3}$. Both asymptotically and in practice the SNFS is much faster than the NFS, with a slowly widening gap: for 1000-bit numbers the SNFS is more than ten thousand times faster, for 1200-bit numbers it is more than 30 thousand times faster.

The function $L(c)$ satisfies various useful but unusual properties, due to the $o(1)$ and $N \to \infty$: $L(c_1)L(c_2) = L(c_1 + c_2)$, $L(c_1) + L(c_2) = L(\max(c_1, c_2))$, and for $c > 0$ and fixed k it is the case that $(\log(N))^k L(c) = L(c)/\log(L(c)) = L(c)$.

2.2 Relation collection

We briefly discuss some aspects of the relation collection step that are relevant for the remainder of the paper and that apply to both the NFS and the SNFS. Let N be the composite to be factored, $c = (64/9)^{1/3}$ (but $c = (32/9)^{1/3}$ if N is special), and assume the proper corresponding d as above. Heuristically it is asymptotically optimal to choose $L(\frac{c}{2})$ as the upper bound for the small factors in the polynomial values and to search for relations among the integer pairs (a, b) with $|a| \leq L(\frac{c}{2})$ and $0 \leq b \leq L(\frac{c}{2})$. For the NFS the rational and algebraic polynomial values then have heuristic probabilities $L(\frac{-c}{8})$ and $L(\frac{-3c}{8})$ to be smooth, respectively; for the SNFS both probabilities are $L(\frac{-c}{4})$. Either way (i.e., NFS or SNFS) and assuming independence of the polynomial values, the polynomial values are both smooth with probability $L(\frac{-c}{2})$. Over the entire search space $L(c)L(\frac{-c}{2}) = L(\frac{c}{2})$ relations may thus be expected, which suffices.

Relation collection can be done using sieving because the search space is a rectangle in $\mathbf{Z}^2$ and because polynomial values are considered. The latter implies that if p divides $g(s)$ (or $f(s)$), then p divides $g(s+kp)$ (or $f(s+kp)$) for any integer k, the former implies that given s all corresponding values $s + kp$ in the search space are quickly located. Thus, for one of the polynomials, sieving is used to locate all pairs in the search space for which the corresponding polynomial value has only factors bounded by $L(\frac{c}{2})$. This costs

$$\sum_{p \text{ prime},\ p \leq L(\frac{c}{2})} \frac{L(c)}{p} = L(c)$$

(for $N \to \infty$, due to the $o(1)$ in $L(c)$) and leads to pairs for which the polynomial value is smooth. Next, in the same way and at the same cost, the pairs are located for which the other polynomial value is smooth. Intersecting the two sets leads to $L(\frac{c}{2})$ pairs for which both polynomial values are smooth.

Sieving twice, once for each polynomial, works asymptotically because $L(c)+L(c) = L(c)$. It may be less obvious that it is also a good approach in practice. After all, after the first sieve only pairs remain that are smooth with respect to the first polynomial, so processing those individually for the second polynomial could be more efficient than reconsidering the entire rectangular search space with another sieve. It will depend on the circumstances what method should be used. For the regular (S)NFS using two sieves is most effective, both asymptotically and in practice: sieving is done twice in a "quick and dirty" manner, relying on the intersection of the two sets to quickly reduce the number of remaining pairs, which are then inspected more closely to extract the relations. In Section 2.4, however, different considerations come into account and one cannot afford a second sieve – asymptotically or in practice – precisely because a second sieve would look at too many values.

As suggested in [29] the sieving task is split up into a large number of somewhat overlapping but sufficiently disjoint subtasks. Given a root z modulo a large prime q of one of the polynomials, a subtask consists of sieving only those pairs (a, b) for which $a/b \equiv z \bmod q$ and for which therefore the values of that polynomial are divisible by q. This implies that the original size $L(c)$ rectangular search space is intersected with an index-q sublattice of $\mathbf{Z}^2$, resulting in a size $L(c)/q$ search space. Sieving can still be used in the new smaller search space, but in a somewhat more complicated manner [29], as first done in [17] and later much better in [13]. Also, more liberal smoothness criteria allow several primes larger than $L(\frac{c}{2})$ in either polynomial value [12]. This complicates the decision of when enough relations have been collected and may increase the matrix size, but leads to a substantial overall speed-up. Another complication that arises is that duplicate relations will be found, i.e., by different subtasks, so the collection of relations must be made duplicate-free before further processing.

2.3 Matrix and filtering

Assume that the numbers of distinct rational and algebraic small primes allowed in the smooth values during relation collection equal r_1 and r_2, respectively. With $r = r_1 + r_2$, each relation corresponds to an r-dimensional vector of exponents. With many distinct potential factors (i.e., large r_1 and r_2) of which only a few occur per smooth value, the exponent vectors are huge-dimensional (with r on the order of billions) and very sparse (on average about 20 non-zero entries).

As soon as $r + 1$ relations have been collected, an even sum of the corresponding r-dimensional vectors (as required to derive a factorization) can in principle be found using linear algebra: with v one of the vectors and the others constituting the columns of an $r \times r$ matrix M_{raw}, an r-dimensional bit-vector x for which $M_{\text{raw}}x$ equals v modulo 2 provides the solution. Although a solution has at least a 50% chance to produce a non-trivial factorization, it may fail to do so, so in practice somewhat more relations are used and more than a single independent solution is derived.

The effort required to find solutions (cf. Section 5) grows with the product of the dimension r and the number of non-zero entries of M_{raw} (the *weight* of M_{raw}). A preprocessing *filtering* step is applied first to M_{raw} in order to reduce this product as much as is practically possible. It consists of a "best effort" to transform, using a sequence of transformation matrices, the initial huge-dimensional matrix M_{raw} of very low average column weight into a matrix M of much lower dimension but still sufficiently low weight. It is not uncommon to continue relation collection until a matrix M can be created in this way that is considered to be "doable" (usage of a second algebraic polynomial for some of our factorizations takes this idea a bit further than usual; cf. sections 3.2 and 4). Solutions for the original matrix M_{raw} easily follow from solutions for the resulting filtered matrix M.

2.4 Coppersmith's factorization factory

Coppersmith, in [8, Section 4], observed that a single linear polynomial g may be used for many different composites as long as their $(d + 1)$st roots are not too far apart, with each composite still using its own algebraic polynomial. Thus smooth $bg(a/b)$-values can be precomputed in a sieving step and used for each of the different factorizations, while amortizing the precomputation cost. We sketch how this works, referring to [8, Section 4] for the details.

After sieving over a rectangular region of $L(2.007)$ rational polynomial values with smoothness bound $L(0.819)$ a total of $L(1.639)$ pairs can be expected (and must be stored for future use) for which the rational polynomial value is smooth. Using this stored table of $L(1.639)$ pairs corresponding to smooth rational polynomial values, any composite in the correct range can be factored at cost $L(1.639)$ per composite: the main costs per number are the algebraic smoothness detection, again with smoothness bound $L(0.819)$, and the matrix step. Factoring $\ell = L(\epsilon)$ such integers costs $L(\max(2.007, 1.639 + \epsilon))$, which is advantageous compared to ℓ-fold application of the regular NFS (at cost $L(1.923)$ per application) for $\ell \geq L(0.084)$. Thus, after a precomputation effort of $L(2.007)$, individual numbers can be factored at cost $L(1.639)$, compared to the individual factorization cost $L(1.923)$ using the regular NFS.

During the precomputation the $L(1.639)$ pairs for which the rational polynomial value is smooth are found by sieving $L(2.007)$ locations. This implies that, from an asymptotic runtime point of view, a sieve should not be used to test the resulting $L(1.639)$ pairs for algebraic smoothness (with respect to an applicable algebraic polynomial), because sieving would cost $L(2.007)$. As a result each individual factorization would cost more than the regular application of the NFS. Asymptotically, this issue is resolved by using the elliptic curve factoring method (ECM, [25]) for the algebraic smoothness test because, for smoothness bound $L(0.819)$, it processes each pair at cost $L(0)$, resulting in an overall algebraic smoothness detection cost of $L(1.639)$. In practice, if it ever comes that far, the ECM may indeed be the best choice, factorization trees ([4] and [15, Section 4]) may be used, or sieving may simply be the fastest option. Because the smooth rational polynomial values will be used by all factorizations, in practice the rational precomputation should probably include, after the sieving, the actual determination of all pairs for which the rational polynomial value is smooth: in the regular (S)NFS this closer inspection of the sieving results takes place only after completing *both* sieves.

These are asymptotic results, but the basic idea can be applied on a much smaller scale too. With a small number ℓ of sufficiently close composites to be factored and using the original NFS parameter choices (and thus a table of $L(1.683)$ as opposed to $L(1.639)$ pairs), the gain approaches 50% with growing ℓ (assuming the matrix cost is relatively minor and disregarding table-storage issues). It remains to be seen, however, if for such small ℓ individual processing is not better if each composite uses a carefully selected pair of polynomials as in [19], and if that effect can be countered by increasing the rational search space a bit while decreasing the smoothness bounds (as in the analysis from [8]).

We are not aware of practical experimentation with Coppersmith's method. To make it realistically doable (in an academic environment) a few suitable moduli could be concocted. The results would, however, hardly be convincing and deriving them would be mostly a waste of computer time – and electric power [21]. We opted for a different approach to gain practical experience with the factorization factory idea, as described below.

2.5 SNFS factorization factory

If we switch the roles of the rational and algebraic sides in Coppersmith's factorization factory, we get a method that can be used to factor numbers that share the same algebraic polynomial, while having different rational polynomials. Such numbers are readily available in the Cunningham project [10, 11, 6][3]. They have the additional advantage that obtaining

[3] On an historical note, the desire to factor the ninth Fermat number $2^{2^9} + 1$, in 1988 the "most wanted" unfactored Cunningham number, inspired the invention of the SNFS, triggering the development of the NFS; the details are described in [23].

their factorizations is deemed to be desirable, so an actual practical experiment may be considered a worthwhile effort. Our choice of target numbers is described in Section 3. First we present the theoretical analysis of the factorization factory with a fixed algebraic polynomial with $O(1)$ coefficients, i.e., the SNFS factorization factory.

Let $L(2\alpha)$ be the size of the sieving region for the fixed shared algebraic polynomial (with coefficient size $O(1)$), let $L(\beta)$ and $L(\gamma)$ be the algebraic and rational smoothness bounds, respectively. Assume the degree of the algebraic polynomial can be chosen as $\delta(\frac{\log(N)}{\log(\log(N))})^{1/3}$ for all numbers to be factored.

The algebraic polynomial values are of size $L[\frac{2}{3}, \alpha\delta]$ and are thus assumed to be smooth with probability $L(-\frac{\alpha\delta}{3\beta})$ (cf. [22, Section 3.16]). With the coefficients of the rational polynomials bounded by $L[\frac{2}{3}, \frac{1}{\delta}]$, the rational polynomial values are of size $L[\frac{2}{3}, \frac{1}{\delta}]$ and may be assumed to be smooth with probability $L(-\frac{1}{3\gamma\delta})$. To be able to find sufficiently many relations it must therefore be the case that

$$2\alpha - \frac{\alpha\delta}{3\beta} - \frac{1}{3\gamma\delta} \geq \max(\beta, \gamma). \tag{1}$$

The precomputation (algebraic sieving) costs $L(2\alpha)$ and produces $L(2\alpha - \frac{\alpha\delta}{3\beta})$ pairs for which the algebraic value is smooth. Per number to be factored, a total of $L(\max(\beta, \gamma) + \frac{1}{3\gamma\delta})$ of these pairs are tested for smoothness (with respect to $L(\gamma)$), resulting in an overall factoring cost

$$L(\max(2\beta, 2\gamma, \max(\beta, \gamma) + \frac{1}{3\gamma\delta}))$$

per number. If $\beta \neq \gamma$, then replacing the smaller of β and γ by the larger increases the left hand side of condition (1), leaves the right hand side unchanged, and does not increase the overall cost. Thus, for optimal parameters, it may be assumed that $\beta = \gamma$. This simplifies the cost to $L(\max(2\gamma, \gamma + \frac{1}{3\gamma\delta}))$ and condition (1) to

$$(2 - \frac{\delta}{3\gamma})\alpha \geq \gamma + \frac{1}{3\gamma\delta},$$

which holds for some $\alpha \geq 0$ as long as $\delta < 6\gamma$. Fixing δ, the cost is minimized when $2\gamma = \gamma + \frac{1}{3\gamma\delta}$ or when $\gamma + \frac{1}{3\gamma\delta}$ attains its minimum; these two conditions are equivalent and the minimum is attained for $\gamma = (3\delta)^{-1/2}$. The condition $\delta < 6\gamma$ translates into

$$\delta < 12^{1/3} \quad \text{respectively} \quad \gamma > 18^{-1/3}.$$

It follows that for δ approaching $12^{1/3}$ from below, the factoring cost per number approaches $L((4/9)^{1/3}) \approx L(0.763)$ from above, with precomputation cost $L(2\alpha)$, $\alpha \to \infty$. These SNFS factorization factory costs should be compared to individual factorization cost $L((32/9)^{1/3}) \approx L(1.526)$ using the regular SNFS, and approximate individual factoring cost $L(1.639)$ after a precomputation at approximate cost $L(2.007)$ using Coppersmith's NFS factorization factory.

Assuming $\gamma = (3\delta)^{-1/2}$, the choices $\gamma = (2/9)^{1/3} \approx 0.606$ and $\alpha = (128/343)^{1/3} \approx 0.808$ lead to minimal precomputation cost $L((4/3)^{5/3}) \approx L(1.615)$, and individual factoring cost $L((4/3)^{2/3}) \approx L(1.211)$. This makes the approach advantageous if more than approximately $L(0.089)$ numbers must be factored (compare this to $L(0.084)$ for Coppersmith's factorization factory). However, with more numbers to be factored, another choice for γ (and thus larger α) may be advantageous, according to a more complete analysis of which we present the conclusion.

Suppose that $\ell = L(\epsilon)$ special numbers must be factored. If $\epsilon < 6^{-1/3}$, then compute γ as the unique positive root of $(3\gamma(\gamma+\epsilon))^2 - 4\gamma - 2\epsilon$ and set $\delta = \frac{3\gamma(\gamma+\epsilon)}{2\gamma+\epsilon}$. Otherwise, if $\epsilon \geq 6^{-1/3}$, then compute γ as the unique positive root of $18\gamma^3\epsilon - 2\gamma - \epsilon$ and set $\delta = \frac{1}{3\gamma^2}$. In either case $\alpha = \frac{3\gamma^2\delta+1}{6\gamma\delta-\delta^2}$ (which simplifies to $\alpha = \frac{2}{6\gamma\delta-\delta^2}$ in the second case) and $\beta = \gamma$ as above. The optimal overall factoring cost is $L(2\alpha)$.

For example, for $\epsilon = 6^{-1/3} \approx 0.550$ we get $\gamma = \epsilon$, $\delta = 2\epsilon$, $\alpha = 3\epsilon/2$, precomputation cost $L(2\alpha) \approx L(1.651)$, and individual factoring cost $L(2\gamma) \approx L(1.101)$. Sets of special numbers can be constructed for which all parameters (including the degree of the shared algebraic polynomial) can be chosen in this way. We leave the construction as an exercise to the reader (for Coppersmith's factorization factory this is trivial).

3 Targets for the SNFS factorization factory

3.1 Target set

For our SNFS factorization factory experiment we chose to factor the Mersenne numbers $2^n - 1$ with $1000 \leq n \leq 1200$ that had not yet been fully factored, the seventeen numbers $2^n - 1$ with $n \in S$ as in the Introduction. We write $S = S_{\text{I}} \cup S_{\text{II}}$, where S_{I} is our first batch containing exponents that are $\pm 1 \bmod 8$ and S_{II} is the second batch with exponents that are $\pm 3 \bmod 8$. Thus

$$S_{\text{I}} = \{1007, 1009, 1081, 1111, 1129, 1151, 1153, 1159, 1177, 1193, 1199\}$$

and

$$S_{\text{II}} = \{1093, 1109, 1117, 1123, 1147, 1171\}.$$

Once these numbers have been factored, only one unfactored Mersenne number with $n \leq 1200$ remains, namely $2^{991} - 1$. It can simply be dealt with using an individual SNFS effort, like the others with $n \leq 1000$ that were still present when we started our project. Our approach would have been suboptimal for these relatively small n.

Around 2009, when we were gearing up for our project, there were several more exponents in the range $[1000, 1200]$. Before actually starting, we first used the ECM in an attempt to remove Mersenne numbers with relatively small factors and managed to fully factor five of them [5]: one with exponent $1 \bmod 8$ and four with exponents $\pm 3 \bmod 8$. Three, all with exponents $\pm 3 \bmod 8$, were later factored by Ryan Propper (using the ECM, [36]) and were thus removed from S_{II}. Some other exponents which were easier for the SNFS were taken care of by various contributors as well, after which the above seventeen remained.

3.2 Polynomial selection for the target set

We used two different algebraic polynomials: $f_{\text{I}} = X^8 - 2$ for $n = \pm 1 \bmod 8$ in S_{I} and $f_{\text{II}} = X^8 - 8$ for $n = \pm 3 \bmod 8$ in S_{II}. This leads to the common roots m_n and rational polynomials g_n corresponding to n as listed in Table 1. Relations were collected using two sieves (one for f_{I} shared by eleven n-values, and one for f_{II} shared by six n-values) and seventeen factorization trees (one for each g_n), as further explained in Section 4. Furthermore,

Table 1. The shared algebraic polynomials, roots, and rational polynomials for the $11 + 6 = 17$ Mersenne numbers $2^n - 1$ considered here.

$f_{\mathrm{I}} = X^8 - 2$

n	$n \bmod 8$	m_n	g_n
1007		2^{126}	$X - 2^{126}$
1111		2^{139}	$X - 2^{139}$
1151	-1	2^{144}	$X - 2^{144}$
1159		2^{145}	$X - 2^{145}$
1199		2^{150}	$X - 2^{150}$
1009		2^{-126}	$2^{126}X - 1$
1081		2^{-135}	$2^{135}X - 1$
1129	1	2^{-141}	$2^{141}X - 1$
1153		2^{-144}	$2^{144}X - 1$
1177		2^{-147}	$2^{147}X - 1$
1193		2^{-149}	$2^{149}X - 1$

$f_{\mathrm{II}} = X^8 - 8$

n	$n \bmod 8$	m_n	g_n
1093		2^{137}	$X - 2^{137}$
1109	-3	2^{139}	$X - 2^{139}$
1117		2^{140}	$X - 2^{140}$
1123		2^{-140}	$2^{140}X - 1$
1147	3	2^{-143}	$2^{143}X - 1$
1171		2^{-146}	$2^{146}X - 1$

$\widetilde{f}_{\mathrm{I}} = X^5 + X^4 - 4X^3 - 3X^2 + 3X + 1$

n	$\widetilde{m}_n$	$\widetilde{g}_n$
1177	$2^{107} + 2^{-107}$	$2^{107}X - (2^{214} + 1)$
1199	$2^{109} + 2^{-109}$	$2^{109}X - (2^{218} + 1)$

in an attempt to reduce the effort to process the resulting matrix, for $n \in \{1177, 1199\}$ additional relations were collected using the algebraic polynomial $\widetilde{f}_{\mathrm{I}}$, as specified in Table 1 along with the common roots $\widetilde{m}_n$ and rational polynomials $\widetilde{g}_n$. Although $n = 1177$ and $n = 1199$ share $\widetilde{f}_{\mathrm{I}}$, to obtain the additional relations it turned out to be more convenient to use the vanilla all-sieving approach from [14] twice, cf. Section 4.4.

Another possibility would have been to select the single degree 6 polynomial $X^6 - 2$. Its relatively low degree and very small coefficients lead to a huge number of smooth algebraic values, all with a relatively large rational counterpart (again due to the low degree). Atypically, rational sieving could have been appropriate, whereas due to large cofactor sizes rational cofactoring would be relatively costly. Overall degree 8 can be expected to work faster, despite the fact that it requires two algebraic polynomials. Degree 7 would require three algebraic polynomials and may be even worse than degree 6 for our sets of numbers, but would have had the advantage that numbers of the form $2^n + 1$ could have been included too

4 Relation collection for the target set

4.1 Integrating the precomputation

The first step of Coppersmith's factorization factory is the preparation and storage of a precomputed table of pairs corresponding to smooth rational polynomial values. With the parameters from [8] this table contains $L(1.639)$ pairs. Assuming composites of relevant sizes, this is huge – possibly to the extent that it is impractical. If we apply Coppersmith's idea as suggested in the second to last paragraph of Section 2.4 to a relatively small number of composites with the original NFS parameter choices, the table would contain $L(1.683)$ pairs, which is even worse.

In our case excessive storage requirements can be avoided. First of all, with the original SNFS parameter choices the table would contain "only" $L(1.145)$ pairs corresponding to smooth algebraic polynomial values, because we are using the factorization factory for the

SNFS with a shared algebraic polynomial. Though better, this is still impractically large. Another effect in our favor is that we are using degree 8 polynomials, which is a relatively large degree compared to what is suggested by the asymptotic runtime analysis: for our N-values the integer closest to $(\frac{3\log(N)}{2\log(\log(N))})^{1/3}$ would be 6. A larger degree leads to larger algebraic values, fewer smooth values, and thus fewer values to be stored.

Most importantly, however, we know our set of target numbers in advance. This allows us to process precomputed pairs right after they have been generated, and to keep only those that lead to a smooth rational polynomial value as well. With ℓ numbers to be factored and $L(\frac{1.523}{2})$ as smoothness bound (cf. Section 2.2), this reduces the storage requirements from $L(1.523)L(\frac{-1.523}{4}) = L(1.145)$ to $\ell L(1.523)L(\frac{-1.523}{2}) = \ell L(0.763)$. For our target sets this is only on the order of TBs (less than six TBs for S_{II}.).

Despite the integration of the algebraic precomputation stage and the processing of the resulting smooth algebraic values on the rational side, the stages are described separately below.

4.2 Algebraic sieving

For the sieving of the polynomial $f_{\mathrm{I}} = X^8 - 2$ from Section 3.2 we used a search space of approximately 2^{66} pairs and varying smoothness bounds. At most two larger primes less than 2^{37} were allowed in the otherwise smooth f_{I}-values.

The sieving task is split up into a large number of subtasks: given a root z of f_{I} modulo a large prime number q, a subtask consists of finding pairs (a, b) for which $a/b \equiv z \bmod q$ (implying that q divides $b^8 f_{\mathrm{I}}(a/b)$) and such that the quotient $b^8 f_{\mathrm{I}}(a/b)/q$ is smooth (except for the large primes) with respect to the largest $h \cdot 10^8$ less than q, with $h \in \{3, 4, 6, 8, 12, 15, 20, 25, 30, 35\}$.

Pairs (a, b) for which $a/b \equiv z \bmod q$ form a two-dimensional lattice of index q in $\mathbf{Z}^2$ with basis $\binom{q}{0}, \binom{z}{1}$. After finding a reduced basis $u, v \in \mathbf{Z}^2$ for the lattice, the intersection of the original search space and the lattice is approximated as $\{\binom{a}{b} = iu + jv : i, j \in \mathbf{Z}, |i| < 2^I, 0 \leq j < 2^J\}$. The bounds $I, J \in \mathbf{Z}_{>0}$ were (or, rather, "are ideally" as this is what we converged to in the course of our experiments) chosen such that $I + J + \log_2(q) \approx 65$ and such that $\max(|a|) \approx \max(b)$, thus taking the relative lengths of u and v into account. Sieving takes place in a size 2^{I+J+1} rectangular region of the (i, j)-plane while avoiding storage for the (even,even) locations, as described in [13]. After the sieving, all f_{I}-values corresponding to the reported locations are divided by q and trial-divided as also described in [13], allowing at most two prime factors between q and 2^{37}. Allowing three large primes turned out to be counterproductive with slightly more relations at considerably increased sieving time or many more relations at the expense of a skyrocketing cofactoring effort.

Each (a, b) with smooth algebraic polynomial value resulting from subtask (q, z) induces a pair $(-a, b)$ with smooth algebraic polynomial value for subtask $(q, -z)$. Subtasks thus come in pairs: it suffices to sieve for one subtask and to recover all smooth pairs for the other subtask before further processing. For $n \geq 1151$ we used most q-values with $4 \cdot 10^8 < q < 8 \cdot 10^9$ (almost 2^{33}), resulting in about 157 million pairs of subtasks. For the other n-values we used fewer pairs of subtasks: about 126 million for $n \in \{1007, 1009\}$ and about 143 million for the others.

Subtasks are processed in disjoint batches consisting of all (prime,root) pairs for a prime in an interval of length 2500 or 10 000. Larger intervals are used for larger q-values, because the latter are processed faster: their sieving region is smaller (cf. above), and their larger smoothness bounds require more memory and thus more cores. After completion of a batch, the resulting pairs are inspected for smoothness of their applicable rational polynomial values as further described below. Processing the batches, not counting the rational smoothness tests, required about **2367** core years. It resulted in $1.57 \cdot 10^{13}$ smooth algebraic values, and thus for each $n \in S_{\mathrm{I}}$ at most twice that many values to be inspected for rational smoothness. Storage of the $1.57 \cdot 10^{13}$ values (in binary format at five bytes per value) would have required 70 TB. As explained in Section 4.1 we avoided these considerable storage requirements by processing the smooth algebraic values almost on-the-fly; this also allowed the use of a more relaxed text format at about 20 bytes per value.

Sieving for $n \in S_{\mathrm{II}}$ was done in the same way. For the polynomial $f_{\mathrm{II}} = X^8 - 8$ and $n \in \{1147, 1171\}$ about 118 million pairs of subtasks were processed for most q-values with $3 \cdot 10^8 < q < 5.45 \cdot 10^9$. For the other n-values in S_{II} about 94% to 96% of that range of q-values sufficed. Overall, sieving for $n \in S_{\mathrm{II}}$ required **1626** core years and resulted in $1.16 \cdot 10^{13}$ smooth algebraic values.

4.3 Rational factorization trees

Each time a batch of f_{I}-sieving subtasks is completed (cf. Section 4.2) the pairs (a, b) produced by it are partitioned over four initially empty queues $\mathcal{Q}_{34}$, $\mathcal{Q}_{35}$, $\mathcal{Q}_{36}$, and $\mathcal{Q}_{37}$: if the largest prime in the factorization of $b^8 f_{\mathrm{I}}(a/b)$ has bitlength i for $i \in \{35, 36, 37\}$ then the pair is appended to $\mathcal{Q}_i$, all remaining pairs are appended to $\mathcal{Q}_{34}$.

After partitioning the new pairs among the queues, the following is done for each $n \in S_{\mathrm{I}}$ (cf. Section 3.1). For all pairs (a, b) in $\cup_{i=34}^{\alpha(n)} \mathcal{Q}_i$, with $\alpha(n)$ as in Table 2, the rational polynomial value $bg_n(a/b)$ (with g_n as in Table 1) is tested for smoothness: if $bg_n(a/b)$ is smooth, then (a, b) is included in the collection of relations for the factorization of $2^n - 1$, else (a, b) is discarded. The smoothness test for the $bg_n(a/b)$-values is conducted simultaneously for all pairs $(a, b) \in \cup_{i=34}^{\alpha(n)} \mathcal{Q}_i$ using a factorization tree as in [15, Section 4] (see also [4]) with $\tau(n) \cdot 10^8$ and $2^{\beta(n)}$ as smoothness and cofactor bounds, respectively (with $\tau(n)$ and $\beta(n)$ as in Table 2). Here the cofactor bound limits the number and the size of the factors in $bg_n(a/b)$ that are larger than the smoothness bound.

For all $n \in S_{\mathrm{I}}$, besides the runtimes Table 2 also lists the numbers of relations found, of free relations [24], of relations after duplicate removal (and inclusion of the free relations), and of prime ideals that occur in the relations before the first singleton removal (where the number of prime ideals is the actual dimension of the exponent vectors). All resulting raw matrices are over-square. For $n \in \{1193, 1199\}$ the over-squareness is relatively small. For $n = 1193$ we just dealt with the resulting rather large filtered matrix. For $n = 1199$, and for $n = 1177$ as well, additional sieving was done, as further discussed in the section below. The unusually high degree of over-squareness for the smaller n-values is a consequence of the large amount of data that had to be generated for the larger n-values, and that could be included for the smaller ones at little extra cost.

Completed batches of subtasks for f_{II}-sieving were processed in the same way. The results are listed in Table 2.

4.4 Additional sieving

In an attempt to further reduce the size of the (filtered) matrix we collected additional relations for $n \in \{1177, 1199\}$ using the degree 5 algebraic polynomial $\widetilde{f_{\mathrm{I}}}$ and the rational poly-

Table 2.

n	$\alpha(n)$	$\tau(n)$	$\beta(n)$	core years	relations found	relations free	relations total unique	occurring prime ideals
1007	34	5	99	26	6 157 265 485	47 681 523	4 083 240 054	1 488 688 670
1009	34	5	99	26	6 076 365 897	47 681 523	4 030 378 014	1 487 997 805
1081	35	5	103	48	7 704 145 069	92 508 436	5 484 250 026	2 828 752 381
1111	35	5	103	46	5 636 554 807	92 508 436	4 045 778 202	2 744 898 588
1129	35	5	103	47	4 860 167 788	92 508 436	3 447 412 400	2 690 405 347
1151	36	5	105	77	9 026 908 346	179 644 953	6 878 035 126	5 229 081 896
1153	36	5	105	78	8 919 329 699	179 644 953	6 798 580 785	5 219 976 433
1159	36	5	105	78	8 494 336 817	179 644 953	6 454 287 572	5 179 538 761
1177	37	20	138	140	15 844 796 536	349 149 710	12 687 801 912	10 098 132 272
1193	37	20	141	171	13 873 940 124	349 149 710	11 120 476 664	9 912 486 202
1199	37	20	141	169	13 201 986 116	349 149 710	10 600 157 337	9 795 656 570
core years for $n \in S_{\rm I}$:				**906**				
1093	35	5	103	37	5 380 284 567	92 508 436	3 777 018 420	2 736 825 054
1109	36	5	105	55	9 621 428 465	179 644 953	7 102 393 219	5 134 440 256
1117	36	5	105	55	8 930 755 992	179 644 953	6 762 813 242	5 220 018 492
1123	36	5	105	54	8 686 858 952	179 644 953	6 567 794 152	5 197 770 153
1147	37	20	138	122	15 404 494 545	349 149 710	12 096 909 112	9 967 719 536
1171	37	20	138	115	12 240 930 101	349 149 710	9 688 750 293	9 556 433 885
core years for $n \in S_{\rm II}$:				**438**				

nomials $\widetilde{g}_n$ from Table 1. These two n-values share $\widetilde{f}_{\rm I}$, so we could have used Coppersmith's approach. For various reasons we treated them separately using the software from [14].

For $n = 1177$ we used on the rational side smoothness bound $3 \cdot 10^8$, cofactor bound 2^{109}, and large factor bound 2^{37}. On the algebraic side these numbers were $5 \cdot 10^8$, 2^{74}, and 2^{37}. Using large primes $q \in [3 \cdot 10^8, 3.51 \cdot 10^8]$ on the rational side (as opposed to the algebraic side above) we found 1 640 189 494 relations, of which 1 606 180 461 remained after duplicate removal. With 1 117 302 548 free relations this led to a total of 2 723 483 009 additional relations. With the 12 687 801 912 relations found earlier, this resulted in 15 411 284 921 relations in total, involving 15 926 778 561 prime ideals. Although this is not over-square (whereas the earlier relation set for $n = 1177$ from Section 4.3 was over-square), the new free relations contained many singleton prime ideals, so that after singleton removal the matrix was easily over-square. The resulting filtered matrix was deemed to be small enough.

For $n = 1199$ the rational smoothness bound is $4 \cdot 10^8$. All other parameters are the same as for $n = 1177$. After processing the rational large primes $q \in [4 \cdot 10^8, 6.85 \cdot 10^8]$ we had 6 133 381 386 degree 5 relations (of which 5 674 876 905 unique) and 1 117 302 548 free relations. This led to 17 392 336 790 relations with 15 955 331 670 prime ideals and a small enough filtered matrix.

The overall reduction in the resulting filtered matrix sizes was modest, and we doubt that this additional sieving experiment, though interesting, led to an overall reduction in runtime. On the other hand, spending a few months (thus a few hundred core years) on additional sieving hardly takes any human effort, whereas processing (larger) matrices is (more) cumbersome. Another reason is that we have resources available that cannot be used for matrix jobs.

4.5 Equipment used

Relation collection for $n \in S_{\text{I}}$ was done from May 22, 2010, until February 21, 2013, entirely on clusters at EPFL as listed in Table 3: 82% on lacal_1 and lacal_2, 12% on pleiades, 3% on greedy, and 1.5% on callisto and vega each, spending 3273 $(2367 + 906)$ core years. Furthermore, 65 and 327 core years were spent on lacal_1 and lacal_2 for additional sieving for $n = 1177$ and $n = 1199$, respectively. Thus a total of **3665** core years was spent on relation collection for $n \in S_{\text{I}}$.

Relation collection for $n \in S_{\text{II}}$ was done from February 21, 2013, until September 11, 2014, on part of the XCG container cluster at Microsoft Research in Redmond, USA, and on clusters at EPFL: 46.5% on the XCG cluster, 45.5% on lacal_1 and lacal_2, 5% on castor, 2% on grid, and 1% on greedy, spending a total of **2064** $(1626 + 438)$ core years. It followed the approach described above for f_{I}, except that data were transported on a regular 500 GB hard disk drive that was sent back and forth between Redmond and Lausanne via regular mail.

Table 3. Description of available hardware. We have 100% access to the equipment at LACAL and to 134 nodes of the XCG container cluster (which contains many more nodes) and limited access to the other resources. A checkmark (✓) indicates InfiniBand network. All nodes have 2 processors.

location	name	processor	nodes	cores per node	cores	GHz	GB RAM per node	GB RAM per core	TB disk space
EPFL	✓bellatrix	Sandy Bridge	424	16	6784	2.2	32	2	
	callisto	Harpertown	128	8	1024	3.0	32	4	
	castor	Ivy Bridge	52	16	832	2.6	50: 64 2:256	4 16	22
	greedy	≈ 1000 mixed cores, ≈ 1 GB RAM per core; 70% windows, 25% linux, 5% mac							
	vega	Harpertown	24	8	192	2.66	16	2	
LACAL	✓lacal_1	AMD	53	12	636	2.2	16	$1\frac{1}{3}$	
	✓lacal_2	AMD	28	24	672	1.9	32	$1\frac{1}{3}$	
	pleiades	Woodcrest	35	4	140	2.66	8	2	
	storage server	AMD	1	24	24	1.9	32	$1\frac{1}{3}$	58
Microsoft Research	part of the XCG container cluster	AMD	134	8	1072	2.1	32	4	
Switzerland	grid	several clusters at several Swiss institutes							

5 Processing the matrices

Although relation collection could be shared among the numbers, the matrices must all be treated separately. Several of them required an effort that is considerably larger than the matrix effort reported in [20]. There a $192\,795\,550 \times 192\,796\,550$-matrix with on average 144 non-zeros per column (in this section all sizes and weights refer to matrices *after* filtering) was processed on a wide variety of closely coupled clusters in France, Japan, and Switzerland, requiring four months wall time and a tenth of the computational effort of the relation collection. So far it was the largest binary matrix effort that we are aware of, in the public domain. The largest matrix done here is about 4.5 times harder.

5.1 The block Wiedemann algorithm

Wiedemann's algorithm. Given a sparse $r \times r$ matrix M over the binary field $\mathbf{F}_2$ and a binary r-dimensional vector v, we have to solve $Mx = v$ (cf. Section 2.3). The minimal polynomial F of M on the vector space spanned by $\{M^0v, M^1v, M^2v, \ldots\}$ has degree at most r. Denoting its coefficients by $F_i \in \mathbf{F}_2$ and assuming that $F_0 = 1$ we have $F(M)v = \sum_{i=0}^{r} F_i M^i v = 0$, so that x follows as $\sum_{i=1}^{r} F_i M^{i-1} v$. Wiedemann's method [34] determines x in three steps. For any j with $1 \leq j \leq r$ the j-th coordinates of the vectors $M^i v$ for $i = 0, 1, 2, \ldots$ satisfy the linear recurrence relation given by the F_i. Thus, once the first $2r + 1$ of these j-th coordinates have been determined using $2r$ iterations of matrix×vector multiplications (Step 1), the F_i can be computed using the Berlekamp-Massey method [26] (Step 2), where it may be necessary to compute the least common multiple of the results of a few j-values. The solution x then follows using another r matrix×vector multiplications (Step 3).

Steps 1 and 3 run in time $\Theta(rw(M))$, where $w(M)$ denotes the number of non-zero entries of M. With Step 2 running in time $O(r^2)$ the effort of Wiedemann's method is dominated by steps 1 and 3.

Block Wiedemann. The efficiency of Wiedemann's conceptually simple method is considerably enhanced by processing several different vectors v simultaneously, as shown in [9, 33]: on 64-bit machines, for instance, 64 binary vectors can be treated at the same time, at negligible loss compared to processing a single binary vector. Though this slightly complicates Step 2 and requires keeping the 64 first coordinates of each vector calculated per iteration in Step 1, it cuts the number of matrix×vector products in steps 1 and 3 by a factor of 64 and effectively makes Wiedemann's method 64 times faster. This *blocking factor* of 64 can, obviously, be replaced by $64t$ for any positive integer t. This calculation can be carried out by t independent threads (or on t independent clusters, [1]), each processing 64 binary vectors at a time while keeping the $64t$ first coordinates per multiplication in Step 1, and as long as the independent results of the t-fold parallelized first step are communicated to a central location for the Berlekamp-Massey step [1].

As explained in [9, 20] a further speed-up in Step 1 may be obtained by keeping, for some integer $k > 1$, the first $64kt$ coordinates per iteration (for each of the t independent 64-bit wide threads). This reduces the number of Step 1 iterations from $2\frac{r}{64t}$ to $(\frac{1}{k} + 1)\frac{r}{64t}$ while the number of Step 3 iterations remains unchanged at $\frac{r}{64t}$. However, it has a negative effect on Step 2 with time and space complexities growing as $(k+1)^{\mu} t^{\mu-1} r^{1+o(1)}$ and $(k+1)^2 tr$, respectively, for $r \to \infty$ and with μ the matrix multiplication exponent (we used $\mu = 3$).

Double matrix product. In all previous work that we are aware of a single filtered matrix M is processed by the block Wiedemann method. This matrix M replaces the original matrix M_{raw} consisting of the exponent vectors, and is calculated as $M = M_{\text{raw}} \times M_1 \times M_2$ for certain filtering matrices M_1 and M_2. For most matrices here, we adapted our filtering strategy, calculated $\widetilde{M_1} = M_{\text{raw}} \times M_1$, and applied the block Wiedemann method to the $r \times r$ matrix M without actually calculating it but by using $M = \widetilde{M_1} \times M_2$. Because Mv can be calculated as $\widetilde{M_1}(M_2 v)$ at (asymptotic) cost $w(M_2) + w(\widetilde{M_1})$ this is advantageous if $r(w(\widetilde{M_1}) + w(M_2))$ is lower than the product of the dimension and weight resulting from traditional filtering. Details about the new filtering strategy will be provided once we have more experience with it.

Error detection. During relation collection no special attention has to be paid to detect errors due to malfunctioning hardware. Correctness of each of the resulting relations can easily be checked, and incorrect ones can, in principle (but see Section 6), simply be removed. Thus, occasional malfunctions do not noticeably affect the efficiency of the relation collection step.

Mishaps during the matrix step, however, need to be detected as a single incorrect bit may render the entire calculation useless – not something one likes to see after a costly calculation that may last months. Traditionally, simple common sense tricks are used that depend on the matrix step used and that allow detection and, if required, rollback to a recent correct state at relatively small additional cost. They are part of factoring "folklore" and normally not explicitly described. For instance, in [2, Section 3.3], where Gaussian elimination was used, spurious dependencies (such as a d-th column consisting of the sum of columns 1 through $d-1$ for regularly spaced d-values) were upfront included in the matrix. For many later factorizations (such as [12]) block Lanczos was used. This generates a sequence of vectors with most of them mutually orthogonal, so an occasional orthogonality check suffices to keep the calculation on track.

For block Wiedemann, we used the following simple method, used since about 2001 by Emmanuel Thomé [32] and in 2002 independently developed by the first author and Jens Franke to deal with frequently flipping bits which went by unnoticed in the floating-point focussed infrastructure they relied upon; for steps 1 and 3 it later appeared in [16]. For a checkpoint distance c and a random vector z we (reliably) precompute $u = (M^T)^c z$. Because the inner product $\langle u, x \rangle$ equals $\langle z, M^c x \rangle$, probably almost all errors can be detected that occurred in Step 1 between the two consecutive checkpoints x and $M^c x$. In Step 3 one can check that the same checkpoints as in Step 1 are computed and one can do a similar, but faster, inner product check as in Step 1 (this was used once, when some files were not copied or written correctly); we do not elaborate. The result of Step 2 can be checked by verifying that certain coefficients are zero in a product of two large matrix-polynomials. The check can be sped up in a simple randomized manner.

5.2 Matrix results

All matrix calculations were done at EPFL on the clusters with InfiniBand network (bellatrix, lacal_1, and lacal_2) and the storage server (cf. Table 3). Despite our limited access to bellatrix, it was our preferred cluster for steps 1 and 3 because its larger memory bandwidth (compared to lacal_1 and lacal_2) allowed us to optimally run on more cores at the same time while also cutting the number of core years by a factor of about two (compared to lacal_1). The matrix from [20], for instance, which would have required about 154 core years on lacal_1 would require less than 75 core years on bellatrix.

Table 4 lists some data for all matrices we processed. Jobs were usually run on a small number of nodes (running up to five matrices at the same time), as that requires the least amount of communication and storage per matrix and minimizes the overall runtime. Extended wall times were of no concern. The Berlekamp-Massey step, for which there are no data in Table 4, was run on the storage server. Its runtime requirements varied from several days to two weeks, using just 8 of the 24 available cores, writing and reading intermediate results to and from disk to satisfy the considerable storage needs. For each of the numbers Step 2 thus took less than one core year.

The error detection methods proved their worth: at least once in Step 1 during the start-up phase of bellatrix and occasionally for lacal_1 and lacal_2 due to writing problems on the network file system.

6 Factorizations

For most n the matrix solutions were processed in the usual way [27, 28, 3] to find the unknown factors of $2^n - 1$. This required an insignificant amount of runtime. The software from [3] is, however, not set up to deal with more fields than the field of rational numbers and a single

Table 4. Data about the matrices processed, as explained in Section 5.1, with $\widetilde{M_1}$, M_2, and M matrices of sizes $r \times \widetilde{r}$, $\widetilde{r} \times (r+\delta)$, and $r \times (r+\delta)$, respectively, for a relatively small positive integer δ. Runtimes in italics are estimates for data that were not kept. Starting from Step 3 for $n = 1151$ a different configuration was used, possibly including some changes in our code, and the programs ran more efficiently. Until $n = 1159$ a blocking factor of 128 was used (so t must be even), for $n \in \{1177, 1193, 1199\} \cup S_{\mathrm{II}}$ it was 64 in order to fit on 16 nodes. The green bars indicate the periods that the matrices were processed, on the green scale at the top. Dates are in the format *yymmdd*.

[121207 ...					core years		... 150109]
n	$r, \widetilde{r}, \delta$ or r, δ (cf. above)	weight(s)	t	k	Step 1	Step 3	start - end
1007	$r =$ 38 986 666 $\widetilde{r} =$ 61 476 801, $\delta =$ 420	201.089r 31.518$\widetilde{r}$	12	3	3.5	*2.6*	121207 - 130106 (30 days)
1009	$r =$ 39 947 548 $\widetilde{r} =$ 64 737 522, $\delta =$ 348	202.077r 36.958$\widetilde{r}$	12	2	*3.9*	*2.6*	130424 - 130610 (47 days)
1081	$r =$ 79 452 919 $\widetilde{r} =$ 122 320 052, $\delta =$ 1624	183.296r 15.332$\widetilde{r}$	16	2	20.3	13.5	130130 - 130311 (41 days)
1111	$r =$ 108 305 368 $\widetilde{r} =$ 167 428 008, $\delta =$ 1018	180.444r 13.887$\widetilde{r}$	24	2	41.8	30.6	130109 - 130611 (154 days)
1129	$r =$ 132 037 278 $\widetilde{r} =$ 204 248 960, $\delta =$ 341	180.523r 13.434$\widetilde{r}$	16	2	64.8	44.4	121231 - 130918 (262 days)
1151	$r =$ 164 438 818 $\widetilde{r} =$ 253 751 725, $\delta =$ 911	174.348r 11.810$\widetilde{r}$	12	2	130.7	38.3	130316 - 131210 (270 days)
1153	$r =$ 168 943 024 $\widetilde{r} =$ 260 332 296, $\delta =$ 1830	169.419r 11.014$\widetilde{r}$	8	2	75.4	43.3	130326 - 131026 (215 days)
1159	$r =$ 179 461 813 $\widetilde{r} =$ 276 906 625, $\delta =$ 1278	174.179r 11.688$\widetilde{r}$	4	2	*87.0*	58.0	130808 - 140207 (184 days)
1177	$r =$ 192 693 549 $\widetilde{r} =$ 297 621 101, $\delta =$ 1043	216.442r 19.457$\widetilde{r}$	4	3	89.3	74.1	140119 - 140525 (127 days)
1193	$r =$ 297 605 781, $\delta =$ 1024	272.267r	6	3	129.5	105.3	131029 - 140819 (295 days)
1199	$r =$ 270 058 949, $\delta =$ 1064	217.638r	6	3	104.8	86.0	140626 - 141211 (169 days)
		core years for $n \in S_{\mathrm{I}}$:			751.0 +	498.7 =	**1249.7**
1093	$r =$ 90 140 482 $\widetilde{r} =$ 138 965 105, $\delta =$ 1854	204.151r 16.395$\widetilde{r}$	8	3	13.4	10.1	140731 - 140912 (44 days)
1109	$r =$ 106 999 725 $\widetilde{r} =$ 164 731 867, $\delta =$ 1662	216.240r 15.976$\widetilde{r}$	8	3	20.3	15.2	140801 - 140919 (50 days)
1117	$r =$ 117 501 821 $\widetilde{r} =$ 182 813 008, $\delta =$ 1894	202.310r 15.638$\widetilde{r}$	6	3	25.5	20.9	140805 - 141121 (109 days)
1123	$r =$ 124 181 748 $\widetilde{r} =$ 192 010 818, $\delta =$ 3225	197.677r 14.222$\widetilde{r}$	4	3	30.9	24.1	140819 - 141220 (124 days)
1147	$r =$ 154 051 173 $\widetilde{r} =$ 237 416 402, $\delta =$ 1170	218.516r 17.141$\widetilde{r}$	6	3	52.8	39.6	141001 - 150107 (99 days)
1171	$r =$ 224 613 073 $\widetilde{r} =$ 349 164 598, $\delta =$ 1252	215.665r 14.602$\widetilde{r}$	6	3	137.0	105.0	140921 - 150109 (111 days)
		core years for $n \in S_{\mathrm{II}}$:			279.9 +	214.9 =	**494.8**

algebraic number field defined by a single algebraic polynomial (in our case f_{I} for $n \in S_{\mathrm{I}}$ and f_{II} for $n \in S_{\mathrm{II}}$). Using this software for $n \in \{1177, 1199\}$, the values for which additional sieving was done for the polynomials $\widetilde{f}_{\mathrm{I}}$ and $\widetilde{g}_n$ from Table 1, would have required a substantial amount of programming. To save ourselves this non-trivial effort we opted for the naive old-fashioned approach used for the very first SNFS factorizations as described in [24, Section 3] of finding explicit generators for all first degree prime ideals in both number fields $\mathbf{Q}(\sqrt[8]{2})$ and $\mathbf{Q}(\zeta_{11} + \zeta_{11}^{-1})$ and up to the appropriate norms. Because both number fields have class number equal to one and the search for generators took, relatively speaking, an insignificant amount of time, this approach should have enabled us to quickly and conveniently deal with these two more complicated cases as well.

For $n = 1177$, however, we ran into an unexpected glitch: the 244 congruences that were produced by the 256 matrix solutions (after dealing with small primes and units) were not correct modular identities involving squares of rational primes and first degree prime ideal generators. This means that the matrix step failed and produced incorrect solutions, or that incorrect columns (i.e., not corresponding to relations) were included in the matrix. Further inspection learned that the latter was the case. It turned out that due to a buggy adaptation to the dual number field case incorrect “relations” containing unfactored composites (due to the speed requirements unavoidably produced by sieving and cofactorization) were used as input to the filtering step. When we started counting the number of bad inputs, extrapolation of early counts suggested quite a few more than 244 bad entries, implying the possibility that the matrix step had to be redone because the 244 incorrect congruences may not suffice to produce correct congruences (combining incorrect congruences to remove the bad entries). We narrowly escaped because, due to circumstances beyond anyone's control [30], the count unexpectedly slowed down and only 189 bad entries were found. This then led to a total of 195 correct congruences, after which the factorization followed using the approach described above. The bug has been fixed, and for $n = 1199$ the problem did not recur.

The seventeen factorizations that we obtained are listed below: n, the lengths in binary and decimal of the unfactored part of $2^n - 1$, factorization date, the lengths of the smallest newly found prime factor, and the factor.

1007 : 843-bit $c254$, Jan 8 2013, 325-bit $p98$:
45664833523052628586495213371442511740075371951182478448819785894752763553620148815526546415896369

1009 : 677-bit $c204$, Jun 12 2013, 295-bit $p89$:
32801629399316220386255938566077541078836238345868341181567256008155638984594836583203447

1081 : 833-bit $c251$, Mar 11 2013, 380-bit $p115$:
1439581090232360306724652721497221475801893594104335706767629109277502599083325989958974577353063372266168702537641

1111 : 921-bit $c278$, Jun 13 2013, 432-bit $p130$:
9401699217426101126085627400537881688668923430306029902665947240112085572850557654128039535064932539432952669653208185411260693457

1129 : 1085-bit $c327$, Sep 20 2013, 460-bit $p139$:
2682863551849463941555012235061302606113919542117141814168219065469741026973149811937861249380857772014308434017285472953428756120546822911

1151 : 803-bit $c242$, Dec 12 2013, 342-bit $p103$:
8311919431039560964291634917977812765997001516444732136271000611174775264337926657343369109100663804047

1153 : 1099-bit $c331$, Oct 28 2013, 293-bit $p89$:
10122360961247873953624190885178888629606889980435179249683524293313230115056983720103793

1159 : 1026-bit $c309$, Feb 9 2014, 315-bit $p95$:
62999265036082335900111964701462000438592932517815660818451881915621154349210038027033309344287

1177 : 847-bit $c255$, May 29 2014, 370-bit $p112$:
2015660787548923454662590205621123886970085761436021592942859847523108465523348455927947279783179798610711213193

1193 : 1177-bit $c355$, Aug 22 2014, 346-bit $p104$:
85227326201314361823893776605433636670217425388311906457714409016049961507516230416822145599757462472729

1199 : 1041-bit $c314$, Dec 17 2014, 252-bit $p76$:
4218108040611917562429786369962714155026225684343947313001103389074302369031

1093 : 976-bit $c294$, Sep 13 2014, 405-bit $p122$:
46116332943436452551540576315696985297990259869411311813222303231047196444160418969946791520558378694863913363980328293449

1109 : 998-bit $c301$, Sep 20 2014, 483-bit $p146$:

$\begin{cases} 24616019211889325200869055218181070587968640061220328510208043399177422314 8 \\ 89351269580238737955397055519355385591659502001884424804253804809798519 \end{cases}$

1117 : 1102-bit $c332$, Nov 24 2014, 259-bit $p78$:

520973969756320610191611683062985619298001156192232208204963188406964166233609

1123 : 1064-bit $c321$, Dec 23 2014, 285-bit $p86$:

$\begin{cases} 4321749374426178250912405171712520218348595194236252887069688897875722874 34 \\ 21469882287 \end{cases}$

1147 : 873-bit $c263$, Jan 10 2015, 378-bit $p112$:

$\begin{cases} 4716610560459008017869143148988463310496868225044779473593358254259917374 16 \\ 2397368176362680597838841461052544854 17 \end{cases}$

1171 : 1134-bit $c342$, Jan 11 2015, 423-bit $p128$:

$\begin{cases} 1234867096283325844280767538326583834762799758794011750619042027360462044 38 \\ 43301292612106209291538933323931056186142998412331913 \end{cases}$

The total cost for the eleven factorizations for $n \in S_{\text{I}}$ was about **4915** core years, with relation collection estimated at 3665 core years, and all matrices in about 1250 core years. The total cost for the six factorizations for $n \in S_{\text{II}}$ was about **2559** core years, with relation collection estimated at 2064 core years, and all matrices in about 495 core years. The total cost for all seventeen factorizations was close to but less than **7500** core years.

Individual factorization using the SNFS would have cost ten to fifteen thousand core years for all $n \in S_{\text{I}}$ and four to six thousand core years for all $n \in S_{\text{II}}$, so overall we obtained a worthwhile saving.

With smallest newly found factors of 76 and 78 decimal digits (of $2^{1199} - 1$ and of $2^{1117} - 1$, respectively) and a largest factor found using the ECM of 83 decimal digits [35], it may be argued that our ECM preprocessing could have been a bit "luckier"; on the bright side, the 254-digit (844-bit) prime factor of $2^{1117} - 1$ may be a new largest prime factor record for SNFS factorizations.

7 Conclusion

We have shown that given a list of properly chosen special numbers their factorizations may be obtained using Coppersmith's factoring factory with considerable savings, in comparison to treating the numbers individually. Application of Coppersmith's idea to general numbers looks less straightforward. Taking the effects into account of rational versus algebraic precomputation (giving rise to many more smooth values) and of our relatively large algebraic degree (lowering our number of precomputed values), extrapolation of the 70 TB disk space estimate given at the end of Section 4.2 suggests that an EB of disk space may be required if a set S of 1024-bit RSA moduli to be factored is not known in advance. This is not infeasible, but not yet within reach of an academic effort. Of course, these excessive storage problems vanish if S is known in advance. But the relative efficiency of current implementations of sieving compared to factorization trees suggests that $|S|$ individual NFS efforts will outperform Coppersmith's factorization factory, unless the moduli get larger. This is compounded by the effect of advantageously chosen individual roots, versus a single shared root.

Regarding the SNFS factorization factory applied to Mersenne numbers, the length of an interval of n-values for which a certain fixed degree larger than our $d = 8$ is optimal, will be larger than our interval of n-values. And, as the corresponding Mersenne numbers $2^n - 1$ will be larger than the ones here, fewer will be factored by the ECM. Thus, we expect that future table-makers, who may wish to factor larger Mersenne numbers, can profit from the approach described in this paper to a larger extent than we have been able to – unless of course better factorization methods or devices have emerged. Obviously, the SNFS factorization factory can be applied to other Cunningham numbers, or Fibonacci numbers, or yet other special numbers. We do not elaborate.

Acknowledgements. We gratefully acknowledge the generous contribution by Microsoft Research to the relation collection effort reported in this paper. Specifically, we want to thank Kristin Lauter for arranging access to the XCG container lab, Lawrence LaVerne for technical support and Michael Naehrig for his assistance in the coordination of the relation collection. We thank Rob Granger for his comments. This work was supported by EPFL through the use of the facilities of its Scientific IT and Application Support Center: the SCITAS staff members have been tremendously helpful and forthcoming dealing with our attempts to process the seventeen matrix steps in a reasonable amount of time. Finally, this work was supported by the Swiss National Science Foundation under grant numbers 200021-119776, 200020-132160, and 206021-128727 and by the project SMSCG (Swiss Multi Science Compute Grid), with computational infrastructure and support, as part of the "AAA/SWITCH – e-infrastructure for e-science" program.

References

1. K. Aoki, J. Franke, T. Kleinjung, A. K. Lenstra, and D. A. Osvik. A kilobit special number field sieve factorization. In K. Kurosawa, editor, *Asiacrypt 2007*, volume 4833 of *LNCS*, pages 1–12. Springer, Heidelberg, 2007.
2. D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland. The magic words are squeamish ossifrage. In *Asiacrypt 1994*, volume 917 of *LNCS*, pages 263–277. Springer, Heidelberg, 1994.
3. F. Bahr. Liniensieben und Quadratwurzelberechnung für das Zahlkörpersieb, 2005. Diplomarbeit, University of Bonn.
4. D. J. Bernstein. How to find small factors of integers. `http://cr.yp.to/papers.html`, june 2002.
5. J. W. Bos, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. Efficient SIMD arithmetic modulo a Mersenne number. In *IEEE Symposium on Computer Arithmetic – ARITH-20*, pages 213–221. IEEE Computer Society, 2011.
6. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr. *Factorizations of* $b^n \pm 1$*,* $b = 2, 3, 5, 6, 7, 10, 11, 12$ *Up to High Powers*, volume 22 of *Contemporary Mathematics*. American Mathematical Society, First edition, 1983, Second edition, 1988, Third edition, 2002. Electronic book available at: `http://homes.cerias.purdue.edu/~ssw/cun/index.html`, 1983.
7. G. Childers. Factorization of a 1061-bit number by the special number field sieve. Cryptology ePrint Archive, Report 2012/444, 2012. `http://eprint.iacr.org/`.
8. D. Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.
9. D. Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.
10. A. J. C. Cunningham and A. E. Western. On Fermat's numbers. *Proceedings of the London Mathematical Society*, 2(1):175, 1904.
11. A. J. C. Cunningham and H. J. Woodall. Factorizations of $y^n \pm 1$, $y = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers. Frances Hodgson, London, 1925.
12. B. Dodson and A. K. Lenstra. NFS with four large primes: an explosive experiment. In D. Coppersmith, editor, *Crypto 1995*, volume 963 of *LNCS*, pages 372–385. Springer, Heidelberg, 1995.
13. J. Franke and T. Kleinjung. Continued fractions and lattice sieving. In *Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS 2005*, 2005. `http://www.hyperelliptic.org/tanja/SHARCS/talks/FrankeKleinjung.pdf`.
14. J. Franke and T. Kleinjung. GNFS for linux. Software, 2012.
15. J. Franke, T. Kleinjung, F. Morain, and T. Wirth. Proving the primality of very large numbers with fastECPP. In D. A. Buell, editor, *Algorithmic Number Theory – ANTS-VI*, volume 3076 of *LNCS*, pages 194–207. Springer, Heidelberg, 2004.

16. W. Geiselmann, A. Shamir, R. Steinwandt, and E. Tromer. Scalable hardware for sparse systems of linear equations, with applications to integer factorization. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 131–146. Springer, Heidelberg, 2005.
17. R. Golliver, A. K. Lenstra, and K. McCurley. Lattice sieving and trial division. In *Algorithmic Number Theory Symposium – ANTS'94*, volume 877 of *LNCS*, pages 18–27, 1994.
18. J. Harrison. Isolating critical cases for reciprocals using integer factorization. In *IEEE Symposium on Computer Arithmetic – ARITH-16*, pages 148–157. IEEE Computer Society, 2003.
19. T. Kleinjung. On polynomial selection for the general number field sieve. *Mathematics of Computation*, 75:2037–2047, 2006.
20. T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In T. Rabin, editor, *Crypto 2010*, volume 6223 of *LNCS*, pages 333–350. Springer, Heidelberg, 2010.
21. A. K. Lenstra, T. Kleinjung, and E. Thomé. Universal security; from bits and mips to pools, lakes – and beyond. In M. Fishlin and S. Katzenbeisser, editors, *Number theory and cryptography*, volume 8260 of *LNCS*, pages 121–124. Springer, Heidelberg, 2013. `http://eprint.iacr.org/2013/635`.
22. A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, pages 673–715. Elsevier and MIT Press, 1990.
23. A. K. Lenstra and H. W. Lenstra, Jr. *The Development of the Number Field Sieve*, volume 1554 of *LNM*. Springer-Verlag, 1993.
24. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. pages 11–42 in [23].
25. H. W. Lenstra Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.
26. J. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.
27. P. Montgomery. Square roots of products of algebraic numbers. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, Proceedings of Symposia in Applied Mathematics, pages 567–571. American Mathematical Society, 1994.
28. P. Q. Nguyen. A Montgomery-like square root for the number field sieve. In *Algorithmic Number Theory Symposium – ANTS III*, volume 1423 of *LNCS*, pages 151–168, 1998.
29. J. M. Pollard. The lattice sieve. pages 43–49 in [23].
30. B. Radford. Why do people see guardian angels?, August 2013. `http://news.discovery.com/human/psychology/why-people-see-guardian-angels-130813.htm`.
31. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.
32. E. Thomé, 2002. Private communication.
33. E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *Journal of Symbolic Computation*, 33(5):757–775, 2002.
34. D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32:54–62, 1986.
35. P. Zimmermann. 50 large factors found by ECM. `http://www.loria.fr/~zimmerma/records/top50.html`.
36. P. Zimmermann. Input file for Cunningham cofactors. `http://www.loria.fr/~zimmerma/records/c120-355`.