The filtering step of discrete logarithm and integer factorization algorithms

Cyril Bouvier

The filtering step of discrete logarithm and integer factorization algorithms

Cyril BOUVIER

Université de Lorraine, CNRS, INRIA, France

Abstract

The security of most current public-key cryptosystems is based on the difficulty of finding discrete logarithms in large finite fields or factoring large integers. Most discrete logarithm and integer factoring algorithms, such as the Number Field Sieve (NFS) or the Function Field Sieve (FFS), can be described in 3 main steps: data collection, filtering and linear algebra. The goal of the filtering step is to generate a small, sparse matrix over a finite field, for which one will compute the kernel during the linear algebra step. The filtering step is mainly a structured Gaussian elimination (SGE). For the current factorization records, billions of data are collected in the first step and have to be processed in the filtering step. One part of the filtering step is to remove heavy rows of the matrix. The choice of the weight function to select heavy rows is critical in order to obtain the smallest matrix possible. In this paper, several weight functions are studied in order to determine which one is more suited in the context of discrete logarithm and factorization algorithms.

Keywords: filtering, linear algebra, structured Gaussian elimination, sparse linear system, factorization, Number Field Sieve, discrete logarithm, Function Field Sieve.
2000 MSC: 15A12

1. Introduction

The difficulty of the discrete logarithm and integer factorization problems is the key of the security of most current public-key cryptosystems. Most of the algorithms solving these problems use the *relation collection method*. The filtering step is a common step of all such algorithms, like the Quadratic Sieve (QS) or the Number Field Sieve (NFS) for the factorization problem, and the Coppersmith algorithm, the Function Field Sieve (FFS) or the Number Field Sieve (NFS-DL) for the discrete logarithm problem. The main goal of the filtering step is to reduce the size of a large sparse matrix over a finite field, constructed

Email address: Cyril.Bouvier@loria.fr (Cyril BOUVIER)

from the data previously collected by the algorithm, in order to being able to compute its kernel.

The main difference between the discrete logarithm and the factorization context is that in the former the matrix is defined over GF(q), with q a prime of 160 to 320 bits, with current standards of security, whereas in the latter the matrix is defined over GF(2).

The filtering step is mostly a structured Gaussian elimination (SGE) [1, 2]. A complete description of the state of the art of the filtering step in the factorization context can be found in Cavallar's thesis [3]. Some integer factorization tools that contain code for filtering are CADO-NFS [4], Msieve [5] and GGNFS [6]. Code for the filtering step in the discrete logarithm context is available in the development version of CADO-NFS.

After giving on overview of relation collection algorithms in the rest of the introduction, and describing in detail the filtering step in Section 2, we will propose, in Section 3, new weight functions for the filtering step and present experiments on three real-case integer factorizations in Section 4 and Section 5 and on three real-case discrete logarithm computations in Section 6. The main contribution of this article is to present several weight functions, including ones previously described in the literature, in an unified formalism and to compare them on real-case factorizations and discrete logarithm computations. With these comparisons, better weight functions as the ones previously used will emerge.

1.1. Overview of the relation collection method

In order to understand the goal of the filtering step, we give a description of the NFS algorithm to illustrate the relation collection method in the factorization context, and a description of the FFS algorithm to illustrate the relation collection method in the discrete logarithm context. The other algorithms based on relation collection differ in some way, but the filtering step remains the same.

1.1.1. The relation collection method for factoring: the example of NFS

For a more detailed description of NFS, see [7].

The goal of the algorithm is to find an equality between two squares modulo the number n that one wants to factor. The algorithm looks for two integers x and y such that $x^2 \equiv y^2 \pmod{n}$.

The NFS algorithm can be described in 3 main steps. A first step (data collection) computes lots of relations of the form

$$\prod_i p_i^{e_i} = \prod_j q_j^{f_j},$$

where the p_i and the q_j represent prime ideals in two distinct number fields. In the following, we will not make the distinction between prime ideals of the two number fields and call them *ideals*.

The goal is to find a subset of all the relations whose product is a square on each side of the equality. This can be rephrased as: for each ideal p, the sum of the exponents of p in all chosen relations must be even.

This can be translated in a linear algebra problem. If one sees the relations as rows of a matrix where each column corresponds to an ideal, the coefficients of the matrix are the exponents of the ideals in the relations. As we are looking for even exponents, one can consider the matrix over GF(2). Finding a linear combination of relations such as every exponent is even is equivalent to computing the (left) kernel of the matrix.

1.1.2. The relation collection method for discrete logarithm: the example of FFS

For a more detailed description of FFS, see [8].

The FFS algorithm computes the logarithm of a element of a finite field by computing lots of logarithms of "small" elements of the field. In order to do so, there is still a data collection step that will compute relations between ideals in two distinct function fields. Then, one wants to compute "logarithms" of these ideals. This can be translated in a linear algebra problem. If, once again, one sees the relations as rows of a matrix where each column corresponds to an ideal, the coefficients of the matrix are the exponents of the ideals in the relations. As we are looking for logarithms in a group of prime order q, one can consider the matrix over GF(q).

Finding the logarithms of the ideals is equivalent to solve this linear system, i.e., computing the (right) kernel of the matrix.

1.1.3. Common points and differences for the filtering step

In the two contexts, the goal of the filtering step is to reduce a matrix defined over a finite field in order to accelerate the linear algebra step, which consists in computing the (left or right) kernel of the matrix and constructing the solution from this kernel.

The *excess* is defined as the difference between the number of rows (relations) and the number of columns (ideals). The *relative excess* is defined as the ratio between the excess and the number of columns (ideals).

In the factorization context, the matrix is defined over GF(2). The linear algebra step computes the left kernel, so the excess should always be positive in order for a nontrivial kernel to surely exist. In practice one tries to have an excess around 150 before linear algebra to have enough vectors in the kernel. In the discrete logarithm context, the matrix is defined over GF(q) where q is a prime of 160 to 320 bits. The linear algebra step computes the right kernel, and in practice one wants this kernel to have dimension 1, so the excess should not be negative. In practice one tries to have an excess of zero before linear algebra as having more rows than columns is useless in this context.

By construction, the matrices at the beginning of the filtering step are sparse (about 20 non-zero coefficients per row). In the linear algebra step, algorithms dedicated to sparse matrices are used in order to deal with matrices from hundreds of thousands to hundreds of millions of rows and columns. The cost of those algorithms depends on the matrix dimensions and on its density. So at the end of the filtering step one should try to keep the matrix as sparse as possible (around 100-200 non-zero coefficients per row in current records).

1.2. Usefulness of the filtering step

The goal of the filtering step is to reduce the size of the matrix that is going to be solved in the linear algebra step. In order to realize in which proportion the reduction is made, we illustrate with data from record factorizations with NFS and record discrete logarithm computations on prime extensions of the binary field with FFS.

The current NFS record factorization is RSA-768 [9], an integer of 768 bits (232 digits). At the beginning of the filtering step, the matrix had about 47 billion rows and 35 billion columns. After the first part of the filtering step (see Section 2 for more detail), the matrix had about 2.5 billion rows and 1.7 billions columns. At the end of the filtering step, the matrix used in the linear algebra had about 193 million rows and columns. So the filtering step reduced the size of the matrix by more than 99% (it was reduced by almost 95% after the first part).

The current FFS record over $\mathrm{GF}(2^n)$, with n prime, is computing a discrete logarithm in $\mathrm{GF}(2^{809})$ [10]. At the beginning of the filtering step, the matrix had about 80 million rows and 40 million columns. After the first part of the filtering step, the matrix had about 10 million rows and columns. At the end of the filtering step, the matrix used in the linear algebra had about 3.6 million rows and columns. So the filtering step reduced the size of the matrix by 96% (it was reduced by almost 88% after the first part).

2. Description of the filtering step

The goal of the filtering step is to construct a matrix which is the smallest and the sparsest possible. Then, in the linear algebra step, one finds the kernel of that matrix. The following description of the state of the art for the filtering step is an adaptation of [3, Chap. 3], in order to work in the factorization and the discrete logarithm context.

The filtering step is classically divided in 3 parts. The first part removes duplicate relations. It will not be studied in this article, as the set of input relations will always be a set of unique relations. The second part, called *purge* (Section 2.1), removes singletons (ideals appearing only once, or equivalently columns with only one non-zero coefficient) and reduces the size of the matrix thanks to the *clique removal* algorithm. The last part, called *merge* (Section 2.2), is a beginning of Gaussian elimination that reduces the size of the matrix by combining rows (and thus making the matrix less sparse).

The purge algorithm is similar to a structured Gaussian elimination (SGE) [1, 2]. SGE begins by removing columns without non-zero coefficients, which do not exist in our case by construction of the matrix. Then, SGE removes columns with exactly one non-zero coefficient and heavy rows, which corresponds to the purge algorithm. The difference between SGE and the filtering step of relation collection algorithms is that purge is followed by merge, so the best weight function adapted to these two cases may be different.

Definition 1. The weight of a row or a column of the matrix is the number of non-zero coefficients of the row or the column. By extension, one can talk about the weight of a relation or an ideal. The total weight of a matrix is the number of non-zero coefficients in the matrix.

2.1. Purge

The purge algorithm is identical in the factorization and the discrete logarithm contexts.

A singleton is a column of the matrix of weight 1 (with only one non-zero coefficient). The corresponding row and column in the matrix can be removed from the matrix without loss of any information. The first part of purge is to remove all singletons and the corresponding rows and columns. When one removes all singletons, the final result does not depend on which order the singletons are removed. Moreover, when one removes a singleton, one removes a row and a column, so the excess does not change (except in the very rare case where a relation contains more than one singleton).

Example 2. *Let us consider the matrix with excess* 2

$$M_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

One can see that the first column is a singleton, so one can remove the first column and the second row. By doing this, one creates a new singleton (the second column of M_0*) and so one can remove the second column and the first row. The remaining matrix is*

$$M_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with no more singleton and still excess 2*.*

Removing excess rows is possible while one is sure that the kernel is still of desired dimension. In the factorization context, when one removes a row, one looses some information on the kernel, but as only a few vectors of the kernel are necessary, one can remove rows until the excess is around 150. In the discrete logarithm context, no information is lost while the excess in non-negative.

In particular, if there is a column of weight 2 and one of the rows corresponding to a non-zero coefficient is removed, then the column becomes a singleton and will be removed during the next singleton phase (as well as the other row corresponding to the other non-zero coefficient). Thus two rows and a column of the matrix will have been deleted, reducing the excess by 1.

Definition 3. Let us consider the graph where the nodes are the rows of the matrix and the edges are the columns of weight 2, connecting the two corresponding rows / nodes. A "clique" is a connected component of this graph.

Remark 4. Even though the component is not a clique in the common sense of graph theory, in the context of filtering it is traditional to call it clique [3], thus we kept that terminology.

Example 5. *Let us consider the matrix M_1 at the end of Example 2. One can see that there are* 3 *cliques: one containing* 3 *rows (the first, second and third rows), one containing two rows (the fourth and the fifth) and one containing only row* 6*. The graph obtained from this matrix is presented in Figure 1. Removing one of these three cliques reduces the excess by* 1*. Removing the clique with three rows connects the two other cliques via the last column.*

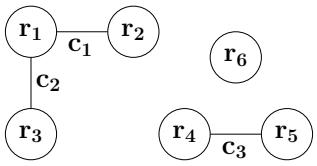

Figure 1: A graph with 3 cliques, constructed from matrix M_1 of Example 2 (r_i corresponds to the ith row and c_j to the jth column).

The stage of purge that removes excess rows is called *clique removal* (or *pruning* as in [3]), as it will, once all the singletons have been removed, compute all the cliques of the matrix and remove some of them. As one can remove any row, one can choose the cliques to remove in order to obtain the smallest, sparsest matrix possible. On the clique removal stage, a weight is given to each clique and the heaviest ones (with respect to that weight function) are deleted. The choice of the weight function associated to a clique is crucial to obtain a small and sparse matrix.

Every algorithm removing rows can be written as a clique removal algorithm with an appropriate weight function, as every row belongs in a clique (possibly a clique with only one row) and removing a row corresponds to removing the entire clique containing this row.

2.2. Merge

The merge stage is a beginning of Gaussian elimination, as one combines some rows and deletes some columns in order to reduce the size of the matrix. In this stage, the total weight of the matrix usually increases.

2.2.1. The merge algorithm in the factorization context

Recall that, in the factorization context, the matrix is defined over GF(2), so that adding two non-zero coefficients gives a zero coefficient.

In order to understand the idea behind the merge algorithm, let us begin with two examples. Let p be an ideal representing a column of weight 2 and let r_1 and r_2 be the two rows corresponding to those two non-zero coefficients. If one replaces in the matrix the row r_1 by $r_1 + r_2$, then the column corresponding to the ideal p becomes a singleton and can be deleted, as well as the row r_2 which is the only remaining row containing the ideal p. The idea is that if the relation r_1 (resp. r_2) is used in the kernel, as the exponent of the ideal p must be even and the only other relation with an odd exponent for p is r_2 (resp. r_1), it must also be used. This operation is called a 2-merge. A 2-merge removes one row and one column (the one corresponding to p) of the matrix, thus leaving the excess unchanged.

Now, if the column corresponding to the ideal p has weight 3 and r_1, r_2 and r_3 are the rows corresponding to those 3 non-zero coefficients, then, as the previous case, if one replaces r_1 by $r_1 + r_3$, r_2 by $r_2 + r_3$, the column corresponding to p becomes a singleton and can be deleted, as well as r_3. This operation is called a 3-merge. A 3-merge removes one row (in this case r_3) and one column (the one corresponding to p) of the matrix, thus leaving the excess unchanged. An important difference with a 2-merge is that there is a choice to be made because there is more than one way of combining the 3 relations. Indeed, one can also choose to add r_2 to both r_1 and r_3 and to delete r_2, or to add r_1 to both r_2 and r_3 and to delete r_1.

Example 6. *For this example, let us consider the matrix M_0 of Example 2, while forgetting that the first column is a singleton. The second column has weight 2, so one can do a 2-merge. If one adds the second row to the first row, then column 2 becomes a singleton and the second column and the second row can be deleted.*

An example of 3-merge comes from the third column which has weight 3. One can see that, in order to minimize the total weight of the matrix, one should add row 4 to row 1 and row 3.

The following definition generalizes those examples to a k-merge, for any integer $k \geq 2$.

Definition 7. Let $k \geq 2$ be a positive integer. Let p be an ideal representing a column of weight k and let $r_1, \ldots, r_k$ be the k rows corresponding to these k non-zero coefficients. A k-merge is a way of performing successive rows additions of the form $r_i \leftarrow r_i + r_j$, with $i \neq j$ and $1 \leq i, j \leq k$, such that the column corresponding to p becomes a singleton that is deleted (as well as the only remaining row containing a non-zero coefficient in that column).

A k-merge removes one row and one column, so a k-merge does not change the excess of the matrix (except in the very rare case where the excess can increase if doing a k-merge allows one to delete more than one column).

A 2-merge always reduces the total weight of the matrix. If the two relations r_1 and r_2 had respectively w_1 and w_2 non-zero coefficients, then the combined row $r_1 + r_2$ will have at most $w_1 + w_2 - 2$ non-zero coefficients. So every 2-merge reduces the number of rows by 1 and the total weight by at least 2. For a k-merge ($k > 2$), the total weight of the matrix will increase in general. In order to control the increase of the total weight of the matrix, one will choose the way of combining the k rows that minimize this increase. In order to do that, one can see the rows as nodes of a graph where all nodes are connected with weights corresponding to the weight of the row resulting of the addition of the two nodes. Finding the best way to combine the k rows is equivalent to finding a spanning tree of minimal weight in this graph (an example is given in Figure 2).

	r_2	r_4	r_5	r_7	r_8
r_1	4	2	4	2	3
r_2		4	2	4	3
r_4			2	2	1
r_5				2	1
r_7					1

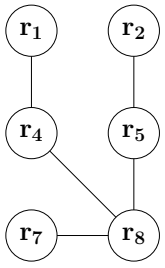

Figure 2: Example of a 6-merge from the sixth column of matrix M_0 of Example 2. The table on the left gives the weight of the sum of r_i and r_j, $i < j$. The figure on the right shows the minimal spanning tree associated to these weights; the edges link two nodes that are going to be added in the 6-merge.

Usually, Markovitz algorithm [11] is used to choose the merges that are going to be made. The merges are in a heap sorted by the weight increase corresponding to that merge.

The merge algorithm always reduces the number of rows but usually increases the total weight of the matrix. So one has to stop while the matrix is still sparse.

2.2.2. Adaptation to the discrete logarithm context

There is only one change is the merge algorithm in the discrete logarithm context. If p is an ideal which occurs in r_i and r_j and if one computes $r_i \leftarrow r_i + r_j$, the column corresponding to p does not usually disappear in the new relation. In order to achieve that, one has to compute $r_i \leftarrow \alpha r_i + \beta r_j$ where α and β are non-zero and depend on the values of the coefficients corresponding to p in r_i and r_j. Apart from that, nothing changes in the merge algorithm.

In theory, the values of the coefficients are in $\mathrm{GF}(q)$, and when two rows are combined the computation of the new coefficients should be done modulo q, but in practice one considers that the coefficients lie in the ring of integers $\mathbb{Z}$. Experimental data show that the initial values of the coefficients are very small relatively to q and that, at the end of merge, more than 99.5% of the coefficients are in $[-10, 10]$ and more than 90% are ± 1.

The problem we address

Computing the (left or right) kernel in the linear algebra step is done with algorithms dedicated to sparse matrices (as the Wiedemann [12] or Lanczos [13] algorithms). These algorithms have a time complexity which is, when one only considers the first term, proportional to the product of the number of rows and the total weight of the final matrix [14, Section 5]. To sum up, the problem of the filtering step is: given as input a set of unique relations, a target excess and a target average row weight for the final matrix, what weight function should be used in the clique removal algorithm in order for the filtering step to produce the smallest matrix possible?

3. Several weight functions for clique removal

What one wants to improve is the size of the matrix at the end of the filtering step (for a given target excess and average row weight). It means that the weight function used in the clique removal stage has to produce the smallest matrix *at the end of merge*, and not necessarily *at the end of purge*. So a good weight function should remove lots of heavy relations but also reduce the total weight of the matrix and create many 2-merges that will reduce the total weight of the matrix during merge.

Before defining the weight functions that are going to be studied, some notations are needed. At the beginning of the clique removal stage, it is assumed that there is no more singleton. The size of the matrix is denoted as N and the total weight is denoted as W (number of non-zero coefficients). Let c be a clique. The number of relations contained in the clique is denoted as $n(c)$. By definition of a clique, the $n(c)$ relations of the clique c are linked by $n(c) - 1$ columns of weight 2. The weight associated to the clique is denoted as $\Omega(c)$. If p is an ideal, then $w(p)$ is the weight of the column that corresponds to the ideal p.

One wants to delete ideals of small weight in order to obtain more singletons and 2-merges, so the weight associated to a column appearing in the clique should be a decreasing function of $w(p)$. Also, due to the nature of the data collecting step, large ideals have very few occurrences, so the weight associated to a column appearing in the clique could also be an increasing function of the "size" of p.

3.1. Weight functions studied

We propose to consider weight functions of the following form: $\Omega(c) = \Lambda(c) + \nu(c)$.

The function Λ measures the contribution of the ideals of weight greater or equal to 3. This function will favor removing cliques with many ideals, or cliques which contain ideals that can create 2-merges. In the experiments, seven functions $\Lambda_0, \ldots, \Lambda_6$ are tried, they all are of the following shape:

$$\Lambda_i(c) = \sum_{p \in c, w(p) \geq 3} \lambda_i(w(p)),$$

where the sum is taken for all ideals p that appear in a relation of the clique and with weight greater or equal to 3 (an ideal appearing in more than one relation of the clique contributes for each appearance). The formulae for the λ_i functions are:

$$\lambda_0(w) = 1, \quad \lambda_1(w) = \left(\frac{2}{3}\right)^{w-2}, \quad \lambda_2(w) = \left(\frac{1}{2}\right)^{w-2}, \quad \lambda_3(w) = \left(\frac{4}{5}\right)^{w-2},$$
$$\lambda_4(w) = \frac{1}{\log_2(w)}, \quad \lambda_5(w) = \frac{2}{w}, \quad \text{and} \quad \lambda_6(w) = \frac{4}{w^2}$$

Note that for all $\lambda_0, \ldots, \lambda_6$ functions, the value in $w = 2$ is 1. The case where the Λ function is zero is treated in the additional weight functions described below.

The function ν measures the contribution of the ideals of weight 2, that is the ideals that link the relations of the clique together. So the function ν is in fact a function depending of $n(c)$ and will try to remove cliques with many relations in order to reduce the size of the matrix. The following formulae for the ν function were tried: $\nu_0(c) = 0$, $\nu_1(c) = \frac{n(c)}{4}$, $\nu_2(c) = \frac{n(c)}{2}$ and $\nu_3(c) = n(c)$.

With these seven formulae for Λ and four formulae for ν, one can construct twenty-eight different weight functions $\Omega_{xy}(c) = \Lambda_x(c) + \nu_y(c)$. For example,

$$\Omega_{23}(c) = \Lambda_2(c) + \nu_3(c) = \sum_{w(p)\geq 3} \frac{1}{2^{w(p)-2}} + n(c).$$

Remark 8. In the discrete logarithm context, the weight functions do not depend on the values of the non-zero coefficients but depend only, *via* $w(p)$, on the fact that the coefficients are zero or non-zero.

In addition to the 28 weight functions constructed above, 3 others were tried:

- The relations of the clique are linked with columns of weight 2 that can be used to perform 2-merges. The weight of the clique c is the weight of the remaining relation once all the possible 2-merges inside the clique have been made. The weight of the remaining relation is exactly the number of ideals that appear an odd number of times in the clique, so

  $$\Omega_{s0}(c) = \sum_{w(p)\geq 2} \frac{1 - (-1)^{w_c(p)}}{2},$$

  where $w_c(p)$ counts the number of relations belonging to the clique c that contain the ideal p. This function is similar to Ω_{00}, if one considers that $w_c(p) = 0$ or 1 when $w(p) \geq 3$ (which is almost always the case for "large" ideals).

- The weight function corresponds to the case where the Λ function is zero:

  $$\Omega_{s1}(c) = \nu_3(c) = n(c).$$

- The expression of this weight function is inspired from the derivative of the product $N \times W$, which is what we are trying to minimize:

$$\Omega_{s2}(c) = \frac{n(c)}{N} + \frac{\Lambda_0(c)}{W}.$$

In total, 31 weight functions were tested, the results of their comparison are given in Section 4 in the context of integer factorization and in Section 6 in the context of discrete logarithm.

3.2. Previously used weight functions

In Cavallar's thesis, a weight function is proposed by the author [3]:

> "The metric being used weighs the contribution from the small prime ideals by adding 1 for each relation in the clique and 0.5 for each free relation. The large prime ideals which occur more than twice in the relation table contribute 0.5^{f-2} where f is the prime ideal's frequency."

Since the free relations only represent a small percentage of the total amount of relations (less than 0.01% for RSA-768), one can considered that the weight function Ω_{23} is the same as the one defined by Cavallar.

The weight function used in Msieve 1.50 is Cavallar's weight function[1], similar to Ω_{23}, where only ideals up to weight 15 are considered. In practice, it gives exactly the same results. Cavallar's weight function was also used in the filtering code of the CWI factoring tool.

In GGNFS 0.77.1, the weight of a clique is the sum of the memory used to store all the relations of the clique, which correspond, up to a constant, to Ω_{03}.

The weight function Ω_{s1} is, up to a constant, the default weight function used in CADO-NFS 1.0 and CADO-NFS 1.1. The weight function Ω_{s2} was also available in the source code of CADO-NFS 1.0 and CADO-NFS 1.1 as an alternative to Ω_{s1}.

4. Comparison of the weight functions in a NFS setup

The 31 weight functions described in Section 3 have been implemented in the development version of CADO-NFS and have been tested on data from three different real-case factorizations: RSA-155, B200 and RSA-704. The original data were used for B200 and RSA-704, and the data for RSA-155 were recomputed with CADO-NFS.

RSA-155 is a 155-digit, 512-bit integer that was factored in 1999 [15] using the NFS algorithm. It was part of the RSA challenge. B200 is the 200th Bernoulli number. In the factorization of the numerator of B200, a 204-digit

[1] In Msieve 1.51 the weight function Ω_{50} is used, due to the first version of this article.

(677-bit) composite integer remained unfactored until August 2012 [16]. RSA-704 is a 212-digit, 704-bit integer that was factored in July 2012 using CADO-NFS [17]. It was part of the RSA challenge.

The computations were done with the development version of the CADO-NFS software. As CADO-NFS 1.1, Msieve 1.50 and GGNFS 0.77.1 have different methods of combining the singleton and clique removal stages, the implementation in the development version of CADO-NFS of their weight functions (respectively Ω_{s1}, Ω_{23} and Ω_{03}) allows us to have a fair comparison where only the weight functions differ (and not the interleaving of the singleton removal and clique removal stages).

4.1. Experiments

The purge algorithm used in these computations interleaves 50 stages of singleton removal and 50 stages of clique removal. On each singleton removal stage all the singletons are removed from the matrix. On each clique removal stage, the excess is decreased by 1/50 of the difference beetwen the initial excess and the desired final excess. This strategy is used in the development version of CADO-NFS. The filtering step (purge and merge) was run for the 31 weight functions without any difference in the other parameters, shown in Table 1.

		RSA-155	B200	RSA-704
Before purge	Number of relations	97.0M	702M	833M
	Number of ideals	89.7M	667M	745M
After first singleton removal	Number of relations	62.8M	453M	650M
	Number of ideals	51.9M	385M	552M
	Excess	10.8M	67.4M	98.5M
	Relative excess	20.8%	17.5%	17.9%
End of purge	Excess	160	160	160
End of merge	k-merge for $k \in$	$[2, 30]$	$[2, 30]$	$[2, 50]$
	W/N	100	120	187

Table 1: Individual parameters for each factorization

Let us recall that the time of the linear algebra step is mostly proportional to the product $N \times W$. And the average weight of a row at the end of merge is the same for all weight functions, thus the product $N \times W$ depends only on the size of the matrix at the end of merge. So the smaller the size of the matrix is at the end of merge, the better the weight function is. In the following, when we say that a weight function is $x\%$ better than another, we mean that the product $N \times W$ is $x\%$ smaller, and not only the size of the matrix.

4.2. Results

In this section, only partial results are presented, the complete data are in the Appendix. The only weight functions that are shown are: the two best ones, the one that gives the smallest matrix after purge, the worst one and those of

Cavallar (Ω_{23}), CADO-NFS 1.1 (Ω_{s1}) and GGNFS 0.77.1 (Ω_{03}) if they are not already in the table.

Summaries of the results for RSA-155, B200 and RSA704 are given, respectively, in Table 2, Table 3 and Table 4. Complete data are shown, respectively, in Table A.10, Table A.11 and Table A.12 in the Appendix.

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 11	20801141	6576610	4.33e+15	
(second best) 21	20812812	6586358	4.34e+15	+0.30%
(best after purge) 23	20300078	6689249	4.47e+15	+3.45%
03	20737766	6841292	4.68e+15	+8.21%
(worst) s1	20560066	6961546	4.85e+15	+12.05%

Table 2: Results for RSA-155, see Table A.10 for complete data

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 11	154087021	44202262	2.34e+17	
(second best) 21	153346744	44237278	2.35e+17	+0.16%
(best after purge) 23	150426986	45029911	2.43e+17	+3.78%
03	153261960	45536067	2.49e+17	+6.13%
(worst) s1	154029290	47030544	2.65e+17	+13.21%

Table 3: Results for B200, see Table A.11 for complete data

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 11	312721893	81239661	1.23e+18	
(second best) 10	323519043	81263553	1.23e+18	+0.06%
(best after purge) 23	305215508	82669415	1.28e+18	+3.55%
03	312257402	83835119	1.31e+18	+6.49%
(worst) s1	307415189	85540465	1.37e+18	+10.87%

Table 4: Results for RSA-704, see Table A.12 for complete data

4.3. Interpretation of the results

In the three cases, the best weight function for the whole filtering step (purge and merge) is Ω_{11}. In all cases, Ω_{11} produces a matrix around 3.5% better than the one produced by Cavallar weight function Ω_{23}, 6 to 8% better than the one produced by GGNFS 0.77.1 weight function Ω_{03}, and 11 to 13% better than the one produced by CADO-NFS 1.1 weight function Ω_{s1} (which is also the worst one in all cases).

These three experiments show that the best weight function Ω_{11} for the whole filtering step (purge and merge) is not the one that produces the smallest matrix at the end of purge, which is, in all cases, Cavallar's weight function Ω_{23}.

The matrices produced with Ω_{00} and Ω_{s0} are almost the same, confirming the fact that almost always, if an "large" ideal p of weight $w(p) \geq 3$ appears in a clique, then it appear only once.

The weight functions that have few or no contribution from the number of relations in the clique (or equivalently ideals of weight 2) are the best ones (e.g. Ω_{11}, Ω_{21}, Ω_{10}, Ω_{30}, ...). This means that the number of relations in the clique should have a small contribution in the weight of the clique. For RSA-155, others weight functions of the form $\Delta_1(c) + \alpha\nu_1(c)$ and $\Delta_3(c) + \alpha\nu_1(c)$ were tried with several values of α in $[0, 1]$. For the Δ_3 case, the best matrice is obtained for $\alpha = 0$ (i.e. Ω_{30}). For the Δ_1 case, the best matrice is obtained for a value of α different from 0 and 1 but is less than 0.1% better than the matrice produced by Ω_{11}.

5. Other experiments with RSA-155

5.1. Statistics on weight of ideals and size of cliques

In order to try to explain why weight functions with a small or no contribution from the number of relations in the clique (or equivalently ideals of weight 2) are better, some data on the number of relations in cliques and on the number of ideals of weight 2, 3 and 4 are presented for Ω_{11}, Ω_{23}, Ω_{s1} and Ω_{03} for RSA-155.

Remember that the weight function Ω_{s1} removes cliques with the largest number of relations, and Ω_{03} and Ω_{23} have an important contribution from ideals of weight 2 (or equivalently number of relations in the clique) whereas Ω_{11} has only a small contribution from the number of relations in the clique.

	Number of	Ω_{11}	Ω_{23}	Ω_{s1}	Ω_{03}
At the beginning of purge	ideals of weight 2		18214716		
	ideals of weight 3		9856626		
	ideals of weight 4		5208195		
	cliques with $n(c) \in [1, 5]$		44104433		
	cliques with $n(c) \in [6, 10]$		407970		
	cliques with $n(c) > 10$		30664		
At the last clique removal stage of purge	ideals of weight 2	6.9M	5.7M	4.8M	5.4M
	ideals of weight 3	3.3M	3.7M	4.6M	4.4M
	ideals of weight 4	2.1M	2.3M	2.6M	2.5M
	cliques with $n(c) \in [1, 5]$	14M	15M	16M	15M
	cliques with $n(c) \in [6, 10]$	18k	3.1k	2.5k	5.0k
	cliques with $n(c) > 10$	9	2	4	2

Table 5: Statistics on weight of ideals and size of cliques at the first and last stages of purge for RSA-155.

Data in Table 5 show that even if the weight functions do not take into account the number of relations in the cliques, the cliques with many relations are deleted. The major difference between these weight functions is the number of possible 2-merges at the beginning of the merge algorithm. The gain in terms of number of relations at the end of purge, for Ω_{23} for example, is not worth the loss in terms of the number of 2-merges available at the beginning of merge.

A clique with a large number of relations will anyway be reduced in one single row during merge while decreasing the total weight of the matrix. So it is more important, during the clique removal, to create many 2-merges, or at least k-merges with k small. This explains why the best weight functions are the ones with a small or no contribution from the number of relations in the cliques.

5.2. Influence of the excess

In this experiment, we want to see what is the effect of the initial excess on the size of the final matrix, for different weight functions. Figure 3 shows the value of the product $N \times W$ for the final matrix of RSA-155 depending on the number of unique relations at the beginning, for the following weight functions: Ω_{11} (the best one), Ω_{23} (Cavallar), Ω_{s1} (CADO-NFS 1.1), Ω_{03} (GGNFS 0.77.1) and Ω_{31}.

Figure 3 shows that the best weight function may change with the excess, as shown by the fact that Ω_{31} produces better matrices that Ω_{11} when the relative excess is greater than 63% but worse matrices otherwise.

Figure 3 also shows that the more relations there are the better the weight function Ω_{11} is, relatively to the others (except Ω_{31}, as discussed above). For example, the difference between Ω_{11} and Ω_{23} is less than 0.2% when the relative excess is 0.6% but can go as high as 20% when the relative excess is 133%. This shows that the choice of deleted cliques made with Ω_{11} is better, as the more excess there is, the better Ω_{11} is.

Finally, Figure 3 shows that most of the time the more excess there is, the better the matrix is. The weight function Ω_{s1} is a counterexample as, when the relative excess go past 60%, the matrix gets bigger as the relative excess increases. Further experiments show that it is due do the fact that when the excess is too big, some cliques with only one relation are removed. As the weight function Ω_{s1} depends only on the size of the cliques, some very light relations can be removed which leads to heavier merges.

6. Comparison of the weight functions in a FFS setup

6.1. Experiments and results

The 31 weight functions described in the Section 3 have been implemented in the development version of CADO-NFS and have been tested on data from three computations of discrete logarithms in GF(2^n) with $n = 619$, $n = 809$ and $n = 1039$. The relations used as inputs in these three cases were computed

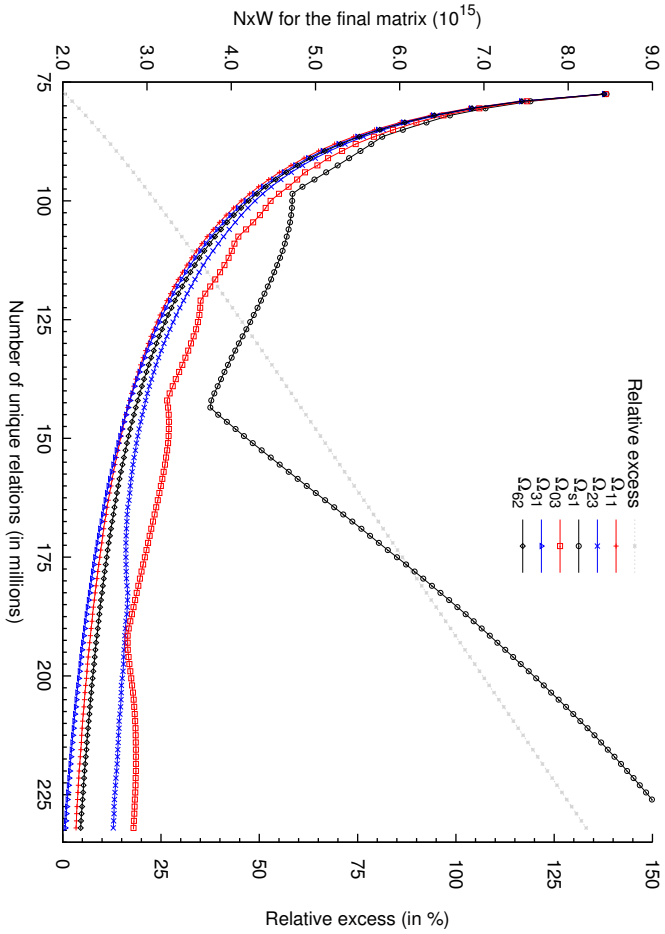

Figure 3: Comparison of weight functions with increasing relative excess

with the relation collection tools for FFS available in the development version of CADO-NFS [18].

For more details on the computation of discrete logarithms in GF(2^{619}), see [19]. For more details on the GF(2^{809}) case, see [10]. In the case of GF(2^{1039}), the complete computation is not finished yet as the linear algebra step has not been done.

The computations of the filtering step were done with the development version of the CADO-NFS software for the 31 weight functions as described in Section 4 with the parameters as shown in Table 6.

As in Section 4, only partial results are presented in this section in Table 7, Table 8 and Table 9. Complete data are shown in the Appendix in Table A.13, Table A.14 and Table A.15.

$\mathrm{GF}(2^n)$ with $n =$		619	809	1039
Before purge	Number of relations	20.5M	81.0M	1306M
	Number of ideals	10.5M	39.4M	986M
After first singleton removal	Number of relations	19.7M	79.1M	1080M
	Number of ideals	9.7M	37.5M	746M
	Excess	10M	41.6M	334M
	Relative excess	103%	111%	44.7%
End of purge	Excess	0	0	0
End of merge	k-merge for $k \in$	$[2, 30]$	$[2, 30]$	$[2, 30]$
	W/N	100	100	100

Table 6: Individual parameters for each discrete logarithm computation

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 31	1978752	648476	4.21e+13	
(second best) 30	2083125	650626	4.23e+13	+0.66%
(best after purge) 62	1893132	671975	4.52e+13	+7.38%
23	1934876	718629	5.16e+13	+22.81%
03	2483373	887541	7.88e+13	+87.32%
(worst) s1	2548952	949242	9.01e+13	+114.27%

Table 7: Results for discrete logarithm in $\mathrm{GF}(2^{619})$, see Table A.13 for complete data

6.2. Interpretation of the results

The first thing to notice is that the differences between the weight functions are larger than in the NFS setup. As shown in Section 5, this is due to the larger relative excess (45–111% against 17–21%).

For the computation in $\mathrm{GF}(2^{1039})$, the results are similar to the results in the NFS case as the relative excess is similar.

For the computations in $\mathrm{GF}(2^{619})$ and $\mathrm{GF}(2^{809})$, the relative excess is larger (more than 100%), so the results are different. The best weight function is Ω_{31} which outperforms Ω_{23} by 20% to 23%, Ω_{03} by 78% to 87% and Ω_{s1} by 115% to 578%. The fact that Ω_{31} is the best weight function when the relative excess is larger is consistent with experiments of Figure 3. The best weight function after purge is not Ω_{23} but Ω_{62}. This is probably due to the large excess as experiments with RSA-155 indicate that it can also be the case in a NFS setup with a lot of excess. The weight function Ω_{23} still produces small matrices after purge but is not competitive for the whole filtering step.

The fact that the best weight function after purge is not the best weight function for the whole filtering step is still true in a FFS setup. The fact that the best weight functions are the ones with few or no contribution from the number of relations in the clique is also still true in a FFS setup.

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 31	9644786	3484569	1.21e+15	
(second best) 30	10126436	3493773	1.22e+15	+0.53%
(best after purge) 62	9153959	3596069	1.29e+15	+6.50%
23	9226205	3822940	1.46e+15	+20.36%
03	12363963	4651334	2.16e+15	+78.18%
(worst) s1	22190421	9075847	8.24e+15	+578.38%

Table 8: Results for discrete logarithm in GF(2^{809}), see Table A.14 for complete data

	After purge	After merge		
Ω	N	N	$W \times N$	
(best) 11	188580425	65138845	4.24e+17	
(second best) 61	185399885	65501515	4.29e+17	+1.12%
(best after purge) 23	182939672	67603362	4.57e+17	+7.71%
03	197703703	74570015	5.56e+17	+31.05%
(worst) s1	203255785	78239129	6.12e+17	+44.27%

Table 9: Results for discrete logarithm in GF(2^{1039}), see Table A.15 for complete data

7. Conclusion

In this article, we presented several weight functions, including ones previously described in the literature, in an unified formalism. Then, we compared them on three real-case factorization and three real-case discrete logarithm computations, using the strategies of CADO-NFS, with the same parameters and where only the weight function differs.

These experiments showed that the best weight functions are the ones with few or no contribution from the ideals of weight 2 (or equivalently from the number of relations in the clique).

We also showed that the best strategy for the purge algorithm only is not the best one for the whole filtering step, this is what makes the difference between the filtering step and Structured Gaussian Elimination (SGE).

New weight functions that outperform previously known ones, in both integer factorization and discrete logarithm context, were identified, which results in a substantial gain in the running time of the linear algebra.

Acknowledgments

The author would like to thank the authors of CADO-NFS and the members of the Caramel team who provided most of the relations used in those experiments. The author would also like to thank Alain Filbois who greatly improve the I/O of the purge program in CADO-NFS, which allows us to do all those experiments in a reasonable time.

References

[1] B. LaMacchia, A. Odlyzko, Solving large sparse linear systems over finite fields, in: A. Menezes, S. Vanstone (Eds.), Advances in Cryptology-CRYPT0' 90, Vol. 537 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1991, pp. 109–133.

[2] C. Pomerance, J. W. Smith, Reduction of huge, sparse matrices over finite fields via created catastrophes, in: Experimental Mathematics, Vol. 1, 1992, pp. 89–94.

[3] S. Cavallar, On the number field sieve integer factorization algorithm, Ph.D. thesis, Universiteit Leiden (2002).

[4] S. Bai, P. Gaudry, A. Kruppa, F. Morain, E. Thomé, P. Zimmermann, Crible algébrique: distribution, optimisation (CADO-NFS), `http://cado-nfs.gforge.inria.fr/`.

[5] J. Papadopoulos, Msieve, `http://sourceforge.net/projects/msieve/`.

[6] C. Monico, GGNFS, `http://www.math.ttu.edu/~cmonico/software/ggnfs`.

[7] A. Lenstra, H. Lenstra, M. Manasse, J. Pollard, The number field sieve, in: A. Lenstra, H. Lenstra (Eds.), The development of the number field sieve, Vol. 1554 of Lecture Notes in Mathematics, Springer Berlin / Heidelberg, 1993.

[8] L. M. Adleman, The function field sieve, in: L. M. Adleman, M.-D. Huang (Eds.), Algorithmic Number Theory, Vol. 877 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1994, pp. 108–121.

[9] 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. T. Riele, A. Timofeev, P. Zimmermann, Factorization of a 768-bit RSA modulus, in: T. Rabin (Ed.), CRYPTO 2010, Vol. 6223 of Lecture Notes in Computer Science, Springer Verlag, Santa Barbara, USA, 2010, pp. 333–350.

[10] R. Barbulescu, C. Bouvier, J. Detrey, P. Gaudry, H. Jeljeli, E. Thomé, M. Videau, P. Zimmermann, Discrete logarithm in GF(2^{809}) with FFS, preprint, 6 pages, available at `http://eprint.iacr.org/2013/197` (2013).

[11] H. Markowitz, The elimination form of the inverse and its application to linear programming, Management Science 3 (3) (1957) 255–269.

[12] D. Wiedemann, Solving sparse linear equations over finite fields, Information Theory, IEEE Transactions on 32 (1) (1986) 54–62.

[13] D. Coppersmith, Solving linear equations over GF(2): block Lanczos algorithm, Linear algebra and its applications 192 (1993) 33–60.

[14] E. Thomé, Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm, in: E. Kaltofen, G. Villard (Eds.), International Conference on Symbolic and Algebraic Computation, ACM, London, Ontario, Canada, 2001, pp. 323–331.

[15] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. Te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffet, C. Putnam, C. Putnam, P. Zimmermann, Factorization of a 512-bit RSA modulus, in: Advances in Cryptology—EUROCRYPT 2000, Springer, 2000, pp. 1–18.

[16] W. Hart, Archives of the Number Theory Mailing List (NMBRTHRY), `https://listserv.nodak.edu/cgi-bin/wa.exe?A0=NMBRTHRY` (2012).

[17] S. Bai, E. Thomé, P. Zimmermann, Factorisation of RSA-704 with CADO-NFS, Cryptology ePrint Archive, Report 2012/369, `http://eprint.iacr.org/` (2012).

[18] J. Detrey, P. Gaudry, M. Videau, Relation collection for the Function Field Sieve, in: P.-M. S. Alberto Nannarelli, P. T. P. Tang (Eds.), ARITH 21 - 21st IEEE International Symposium on Computer Arithmetic, IEEE, 2013, pp. 201 – 210.
URL `http://hal.inria.fr/hal-00736123`

[19] R. Barbulescu, C. Bouvier, J. Detrey, P. Gaudry, H. Jeljeli, E. Thomé, M. Videau, P. Zimmermann, The relationship between some guy and cryptography, ECC2012 rump session talk (humoristic), see `http://ecc.2012.rump.cr.yp.to/` (2012).

Appendix A. Complete Data

The appendix contains the complete data for all the weight functions for RSA-155, B200 and RSA-704, and the computations of discrete logarithms in GF(2^{619}), GF(2^{809}) and GF(2^{1039}). It also contains more information about the parameters used in purge and merge that were not necessary for the understanding of the comparison between the weight functions.

For RSA-155, B200 and RSA-704, during merge, the 32 heaviest columns (with most non-zero coefficients) are buried because they are discarded during the linear algebra step. For the computations of discrete logarithms, no column is buried during merge.

For B200, RSA-704 and the three discrete logarithm computations, the original relations were used, for RSA-155, they were generated with CADO-NFS, with the following polynomials:

$$g(x) = 3216929091619x - 730598439063554685035469587 00$$

$$f(x) = 5256451200x^5 - 408896411797380x^4 + 1161522740010687 9434x^3$$
$$+ 1895260253840098645553615 0x^2 - 3101178076280120395421431 96507x$$
$$- 1232050446355944886746176 17051247 90497$$

For RSA-155, during purge, only ideals greater than 16M on rational side and 32M on algebraic side were taken into account. The complete set of data is shown in Table A.10.

For B200, during purge, only ideals greater than 250M on both sides were taken into account. The complete set of data is shown in Table A.11.

For RSA-704, during purge, only ideals greater than 250M on rational side and 500M on algebraic side were taken into account. The complete set of data is shown in Table A.12.

For GF(2^{619}), during purge, only ideals of degree greater than 19 on both sides were taken into account. The complete set of data is shown in Table A.13.

For GF(2^{809}), during purge, only ideals of degree greater than 23 on both sides were taken into account. The complete set of data is shown in Table A.14.

For GF(2^{1039}), during purge, only ideals of degree greater than 25 on both sides were taken into account. The complete set of data is shown in Table A.15.

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
11	20801141	8.71e+15	6576610	4.33e+15	
21	20812812	8.71e+15	6586358	4.34e+15	+0.30%
61	20591172	8.53e+15	6592609	4.35e+15	+0.49%
10	21677978	9.46e+15	6597020	4.35e+15	+0.62%
30	21066248	8.93e+15	6598208	4.35e+15	+0.66%
12	20525758	8.48e+15	6605260	4.36e+15	+0.87%
50	20940957	8.83e+15	6607958	4.37e+15	+0.96%
60	21672863	9.45e+15	6611637	4.37e+15	+1.07%
31	20732159	8.65e+15	6614847	4.38e+15	+1.17%
22	20437176	8.40e+15	6616157	4.38e+15	+1.21%
51	20632230	8.57e+15	6630136	4.40e+15	+1.63%
32	20570710	8.52e+15	6638018	4.41e+15	+1.88%
62	20377654	8.35e+15	6640471	4.41e+15	+1.95%
40	20855869	8.76e+15	6657267	4.43e+15	+2.47%
52	20493661	8.45e+15	6659152	4.43e+15	+2.53%
13	20361903	8.34e+15	6668262	4.45e+15	+2.81%
41	20628073	8.57e+15	6676570	4.46e+15	+3.06%
33	20433585	8.40e+15	6686196	4.47e+15	+3.36%
23	20300078	8.29e+15	6689249	4.47e+15	+3.45%
42	20514904	8.47e+15	6701793	4.49e+15	+3.84%
63	20321565	8.30e+15	6712215	4.51e+15	+4.17%
53	20400561	8.37e+15	6713693	4.51e+15	+4.21%
43	20434660	8.40e+15	6750781	4.56e+15	+5.37%
20	22876961	1.05e+16	6751559	4.56e+15	+5.39%
02	20858120	8.76e+15	6818390	4.65e+15	+7.49%
01	20860148	8.76e+15	6818411	4.65e+15	+7.49%
s0	21048884	8.92e+15	6831805	4.67e+15	+7.91%
00	21048891	8.92e+15	6831875	4.67e+15	+7.91%
03	20737766	8.65e+15	6841292	4.68e+15	+8.21%
s2	20496848	8.44e+15	6883627	4.74e+15	+9.55%
s1	20560066	8.49e+15	6961546	4.85e+15	+12.05%

Table A.10: Complete data for RSA-155

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
11	154087021	5.23e+17	44202262	2.34e+17	
21	153346744	5.18e+17	44237278	2.35e+17	+0.16%
61	152480272	5.12e+17	44295368	2.35e+17	+0.42%
30	158109764	5.51e+17	44344210	2.36e+17	+0.64%
10	161480932	5.74e+17	44359892	2.36e+17	+0.71%
50	156737711	5.41e+17	44359936	2.36e+17	+0.71%
31	154254136	5.24e+17	44370581	2.36e+17	+0.76%
60	161056386	5.71e+17	44386107	2.36e+17	+0.83%
12	151985472	5.08e+17	44398656	2.37e+17	+0.89%
51	153312434	5.18e+17	44455926	2.37e+17	+1.15%
22	151171217	5.03e+17	44504976	2.38e+17	+1.37%
32	152617177	5.13e+17	44519799	2.38e+17	+1.44%
40	155854758	5.35e+17	44624257	2.39e+17	+1.92%
62	150943964	5.01e+17	44631783	2.39e+17	+1.95%
52	151973891	5.08e+17	44647196	2.39e+17	+2.02%
41	153252776	5.17e+17	44718043	2.40e+17	+2.35%
13	150807090	5.00e+17	44822261	2.41e+17	+2.82%
33	151379703	5.04e+17	44835865	2.41e+17	+2.89%
42	152135118	5.09e+17	44879381	2.42e+17	+3.09%
53	151103756	5.02e+17	44990483	2.43e+17	+3.60%
23	150426986	4.98e+17	45029911	2.43e+17	+3.78%
20	167572608	6.18e+17	45145579	2.45e+17	+4.31%
43	151365155	5.04e+17	45160515	2.45e+17	+4.38%
63	150505303	4.98e+17	45168479	2.45e+17	+4.42%
02	154067402	5.22e+17	45495013	2.48e+17	+5.93%
01	155395404	5.32e+17	45505293	2.48e+17	+5.98%
03	153261960	5.17e+17	45536067	2.49e+17	+6.13%
00	156540087	5.40e+17	45550198	2.49e+17	+6.19%
s0	156540092	5.40e+17	45550331	2.49e+17	+6.19%
s2	151983844	5.08e+17	45897498	2.53e+17	+7.82%
s1	154029290	5.21e+17	47030544	2.65e+17	+13.21%

Table A.11: Complete data for B200

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
11	312721893	2.18e+18	81239661	1.23e+18	
10	323519043	2.34e+18	81263553	1.23e+18	+0.06%
21	312473810	2.18e+18	81379271	1.24e+18	+0.34%
30	317826394	2.26e+18	81411698	1.24e+18	+0.42%
60	324041496	2.34e+18	81418192	1.24e+18	+0.44%
61	309891857	2.14e+18	81427075	1.24e+18	+0.46%
50	316478268	2.24e+18	81480389	1.24e+18	+0.59%
12	308837595	2.13e+18	81584492	1.24e+18	+0.85%
31	312574204	2.18e+18	81589561	1.24e+18	+0.86%
51	311175033	2.16e+18	81724892	1.25e+18	+1.20%
22	307395363	2.11e+18	81774400	1.25e+18	+1.32%
32	309900307	2.14e+18	81854176	1.25e+18	+1.52%
40	315549828	2.22e+18	81971998	1.26e+18	+1.81%
62	306584663	2.10e+18	81998217	1.26e+18	+1.88%
52	308695518	2.13e+18	82055985	1.26e+18	+2.02%
41	311251786	2.16e+18	82176315	1.26e+18	+2.32%
13	306186056	2.09e+18	82336697	1.27e+18	+2.72%
33	307411911	2.11e+18	82425057	1.27e+18	+2.94%
42	309026229	2.13e+18	82465478	1.27e+18	+3.04%
20	335993975	2.52e+18	82558900	1.27e+18	+3.27%
23	305215508	2.08e+18	82669415	1.28e+18	+3.55%
53	306704487	2.10e+18	82735374	1.28e+18	+3.72%
63	305410102	2.08e+18	82943841	1.29e+18	+4.24%
43	307120972	2.10e+18	83091811	1.29e+18	+4.61%
01	315512310	2.22e+18	83775945	1.31e+18	+6.34%
02	314374563	2.21e+18	83783791	1.31e+18	+6.36%
03	312257402	2.18e+18	83835119	1.31e+18	+6.49%
s0	318482486	2.27e+18	83863260	1.32e+18	+6.56%
00	318482490	2.27e+18	83863378	1.32e+18	+6.56%
s2	307155720	2.10e+18	85114020	1.35e+18	+9.77%
s1	307415189	2.11e+18	85540465	1.37e+18	+10.87%

Table A.12: Complete data for RSA-704

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
31	1978752	8.51e+13	648476	4.21e+13	
30	2083125	9.43e+13	650626	4.23e+13	+0.66%
50	1982615	8.55e+13	653358	4.27e+13	+1.51%
32	1938808	8.16e+13	655589	4.30e+13	+2.21%
12	1916454	7.96e+13	657308	4.32e+13	+2.74%
11	1984857	8.54e+13	657967	4.33e+13	+2.95%
61	1926763	8.05e+13	661554	4.38e+13	+4.07%
51	1936270	8.15e+13	663028	4.40e+13	+4.54%
62	1893132	7.77e+13	671975	4.52e+13	+7.38%
22	1906373	7.87e+13	675500	4.56e+13	+8.51%
52	1926009	8.05e+13	677077	4.58e+13	+9.02%
21	1971971	8.42e+13	678108	4.60e+13	+9.35%
60	2125923	9.80e+13	678718	4.61e+13	+9.54%
33	1925359	8.04e+13	678894	4.61e+13	+9.60%
13	1901372	7.83e+13	679181	4.61e+13	+9.69%
10	2182072	1.03e+14	683442	4.67e+13	+11.08%
40	1980510	8.53e+13	692468	4.80e+13	+14.03%
41	1973283	8.46e+13	707211	5.00e+13	+18.94%
20	2227058	1.07e+14	713659	5.09e+13	+21.11%
53	1948261	8.22e+13	714369	5.10e+13	+21.36%
23	1934876	8.09e+13	718629	5.16e+13	+22.81%
42	1975296	8.46e+13	723109	5.23e+13	+24.34%
63	1957756	8.28e+13	737947	5.45e+13	+29.50%
43	1984188	8.52e+13	745938	5.56e+13	+32.32%
03	2483373	1.34e+14	887541	7.88e+13	+87.32%
01	2490814	1.35e+14	887905	7.88e+13	+87.48%
02	2490830	1.35e+14	887956	7.88e+13	+87.50%
s2	2480183	1.34e+14	888767	7.90e+13	+87.84%
s0	2504747	1.37e+14	890097	7.92e+13	+88.40%
00	2504738	1.37e+14	890107	7.92e+13	+88.41%
s1	2548952	1.40e+14	949242	9.01e+13	+114.27%

Table A.13: Complete data for discrete logarithm in GF(2^{619})

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
31	9644786	2.10e+15	3484569	1.21e+15	
30	10126436	2.32e+15	3493773	1.22e+15	+0.53%
50	9761906	2.15e+15	3515248	1.24e+15	+1.77%
32	9424672	2.00e+15	3517593	1.24e+15	+1.90%
12	9313137	1.95e+15	3528628	1.25e+15	+2.54%
11	9670420	2.11e+15	3530193	1.25e+15	+2.64%
51	9454582	2.02e+15	3548367	1.26e+15	+3.70%
61	9360098	1.97e+15	3550649	1.26e+15	+3.83%
62	9153959	1.89e+15	3596069	1.29e+15	+6.50%
52	9336976	1.97e+15	3607460	1.30e+15	+7.18%
22	9238788	1.92e+15	3614219	1.31e+15	+7.58%
60	10265165	2.37e+15	3616670	1.31e+15	+7.73%
33	9284572	1.94e+15	3623881	1.31e+15	+8.16%
21	9544594	2.05e+15	3631009	1.32e+15	+8.58%
13	9159235	1.89e+15	3634841	1.32e+15	+8.81%
10	10569620	2.51e+15	3644690	1.33e+15	+9.40%
40	9838799	2.19e+15	3660460	1.34e+15	+10.35%
41	9634081	2.10e+15	3696603	1.37e+15	+12.54%
42	9535985	2.05e+15	3747073	1.40e+15	+15.63%
53	9298354	1.95e+15	3753044	1.41e+15	+16.00%
20	10617233	2.53e+15	3764939	1.42e+15	+16.74%
23	9226205	1.91e+15	3822940	1.46e+15	+20.36%
43	9477803	2.02e+15	3846167	1.48e+15	+21.83%
63	9238544	1.92e+15	3862733	1.49e+15	+22.88%
s2	12231926	3.38e+15	4649193	2.16e+15	+78.02%
03	12363963	3.46e+15	4651334	2.16e+15	+78.18%
02	12442274	3.50e+15	4652580	2.16e+15	+78.27%
01	12445627	3.50e+15	4652845	2.16e+15	+78.29%
00	12735446	3.67e+15	4703312	2.21e+15	+82.18%
s0	12735441	3.67e+15	4703398	2.21e+15	+82.19%
s1	22190421	1.07e+16	9075847	8.24e+15	+578.38%

Table A.14: Complete data for discrete logarithm in GF(2^{809})

	After purge		After merge		
Ω	N	$W \times N$	N	$W \times N$	
11	188580425	8.49e+17	65138845	4.24e+17	
61	185399885	8.20e+17	65501515	4.29e+17	+1.12%
30	192816556	8.88e+17	65513540	4.29e+17	+1.15%
12	185046190	8.17e+17	65600990	4.30e+17	+1.42%
21	188480429	8.47e+17	65646657	4.31e+17	+1.57%
31	188302437	8.47e+17	65800281	4.33e+17	+2.04%
10	200713508	9.61e+17	65884992	4.34e+17	+2.30%
50	188136760	8.46e+17	65964477	4.35e+17	+2.55%
22	183764221	8.05e+17	66030150	4.36e+17	+2.76%
60	198153114	9.37e+17	66037579	4.36e+17	+2.78%
32	186410816	8.29e+17	66263200	4.39e+17	+3.48%
62	183311722	8.01e+17	66413060	4.41e+17	+3.95%
51	185963237	8.26e+17	66521544	4.43e+17	+4.29%
13	183552821	8.03e+17	66820482	4.46e+17	+5.23%
52	185187044	8.19e+17	67039863	4.49e+17	+5.92%
33	185193044	8.18e+17	67106628	4.50e+17	+6.13%
23	182939672	7.98e+17	67603362	4.57e+17	+7.71%
53	184758976	8.14e+17	67766411	4.59e+17	+8.23%
63	183169307	8.00e+17	67792554	4.60e+17	+8.31%
40	187520013	8.41e+17	68079911	4.63e+17	+9.23%
41	186937457	8.35e+17	68363238	4.67e+17	+10.15%
42	186640959	8.32e+17	68571814	4.70e+17	+10.82%
43	186442611	8.30e+17	68797554	4.73e+17	+11.55%
20	212519429	1.08e+18	69376064	4.81e+17	+13.43%
s2	197233222	9.30e+17	74451493	5.54e+17	+30.64%
03	197703703	9.34e+17	74570015	5.56e+17	+31.05%
01	197944839	9.37e+17	74585164	5.56e+17	+31.11%
02	197940661	9.37e+17	74585212	5.56e+17	+31.11%
00	198444888	9.42e+17	74694800	5.58e+17	+31.49%
s0	198444886	9.42e+17	74694852	5.58e+17	+31.49%
s1	203255785	9.84e+17	78239129	6.12e+17	+44.27%

Table A.15: Complete data for discrete logarithm in GF(2^{1039})