Discrete logarithm in GF(2^{809}) with FFS

The CARAMEL group

Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry,
Hamza Jeljeli, Emmanuel Thomé, Marion Videau, Paul Zimmermann

CNRS, INRIA, Université de Lorraine

April 6, 2013

We give details on solving the discrete logarithm problem in the 202-bit prime order subgroup of $\mathbb{F}_{2^{809}}^\times$ using the Function Field Sieve algorithm (FFS).

To our knowledge, this computation is the largest discrete logarithm computation so far in a binary field extension of prime degree. The Function Field Sieve is the traditional approach for solving these problems, and has been used in previous records for such fields, namely $\mathbb{F}_{2^{619}}$ [3] and $\mathbb{F}_{2^{613}}$ [8].

One should note that an adaptation of the newer $L(1/4 + o(1), \cdot)$ algorithm by Joux [7] also applies to computations of this kind. Presently, the crossover point between the Function Field Sieve and this newer algorithm is not known, and the present computation contributes to giving an idea of the present state of the art of what may be computed using the Function Field Sieve.

Most of the software used for this computation is freely available as part of the `cado-nfs` software suite [1] (although `cado-nfs` originally focuses on the Number Field Sieve, recent additions cover FFS as well).

Various improvements over the different steps of the algorithm are covered in preprints by some of the authors of the present computation [2, 4, 5, 6]. We therefore keep this report very short and refer the interested reader to these articles for more detail.

Discussion on the subgroup considered. We have chosen to attack the discrete logarithm problem in a subgroup which is not $\mathbb{F}_{2^{809}}^\times$ itself, but rather one of its prime-order subgroups, namely the subgroup of prime order ℓ, where ℓ is the 202-bit prime factor of $2^{809} - 1$ given as:

$$\ell = 4148386731260605647525186547488842396461625774241327567978137.$$

The other factor of $2^{809} - 1$ is also prime, and 608-bits long.

The motivation for our choice is related to the cryptographic applications, where the discrete logarithm problem is to be solved only in a subgroup whose size is enough to resist the Pollard Rho attack. We recall, as a comparison, that the original DSA (digital signature algorithm) setup recommends a 160-bit prime order subgroup in the multiplicative group of a 1024-bit finite field. Here, the subgroup chosen is rather over-sized than under-sized, given the expected difficulty of the Pollard Rho attack on a 202-bit group.

Extrapolations from the hardness of our discrete logarithm computation to the hardness of the full discrete logarithm computation in the group $\mathbb{F}_{2^{809}}^\times$ are easy to obtain, as one can get satisfactory estimates by keeping most timings unchanged, and scaling the linear algebra cost by roughly a 4-fold linear factor (the complexity of the linear algebra is dominated by the cost of multiplying an integer modulo ℓ by a word-size integer).

Notation. For compactness, a polynomial of $\mathbb{F}_2[t]$ is represented by the integer obtained when setting $t = 2$ in the polynomial, and we write this integer in hexadecimal so that sparseness is still visible. For instance, `0x11` is $t^4 + 1$, and `0xb00001` is $t^{23} + t^{21} + t^{20} + 1$.

1 A few words on each step

1.1 Polynomial selection

We used the criteria defined in [2] to select the polynomials. It appears that a polynomial $f(x)$ of degree 6 was the best choice, so we computed the α value (a quantity that is similar as the one used when estimating the quality of NFS polynomials) for all the irreducible polynomials of degree 6 in x for which the degree in t of the coefficient of x^i is at most $12-2i$. It was done with a sieving procedure. Among the few polynomials that were preselected for having an α value around -6, only one had a structure that imposes a factor of degree at most one in the resultant of $f(x)$ with a linear monic polynomial $g(x)$. For this reason we chose that one, since 809 is congruent to 5 mod 6. No special care was taken for choosing $g(x)$: a linear monic polynomial with a sparse constant coefficient of degree 135, that we took rather for convenience in writing but this feature of $g(x)$ did not make any difference in the computation. Finally, we chose the following polynomial pair:

$$\begin{aligned} f(x) &= x^6 + \texttt{0x7}x^5 + \texttt{0x6b}x^3 + \texttt{0x1ab}x^2 + \texttt{0x326}x + \texttt{0x19b3} \\ g(x) &= x + \texttt{0x800000000000000000000000000001e7eaa}\ . \end{aligned}$$

The resultant of $f(x)$ and $g(x)$ is $(t+1)$ times an irreducible factor $\varphi(t)$ of degree 809 that we take as a defining polynomial for $\mathbb{F}_{2^{809}}$:

$$\begin{aligned} \varphi(t) = {} & \texttt{0x3fffffffffffffffffffffffffffffffffe8000000000000000000000} \\ & \texttt{cc0cfaeb0e0000000000000000000000000000004dfffffffffffffc3c} \\ & \texttt{3ffc3c3c3fce3c2133ffffffffffffe9697fe96804c84c97e0b0000c00cf} \\ & \texttt{9b0f7675354e79f4cf7c97e29}\ . \end{aligned}$$

This choice of polynomials was driven solely by the efficiency of the relation collection. The genus of the curve corresponding to $f(x,t)=0$ is 19, which is not especially low. The class number is 2073600, which is rather large, and there are some singular points. The only ones that we have to take care of for the computation are the ones at $(x,t)=(\omega,\omega)$, where $\omega^2+\omega+1=0$, which splits into 2 places, and of course its conjugate, which also splits into 2 places. All these complications have essentially no influence on the running time but require some care in the implementation.

1.2 Relation collection

The relation collection was performed using the implementation described in [5]. This is a rather classical sieving method using lattice-sieving for various special-q. We actually ran the relation collection step for two different sets of parameters, in order to compare and be able to see how the tuning of this phase influences the filtering and linear algebra step.

The terminology used in [5] is the classical one, where "factor base bound" is the limit for the degree of the irreducible polynomials that are sieved, and "large prime bound" is the limit for the degree of the polynomials allowed in a relation. These notions are very similar to the ones used when sieving relations for factoring integers using NFS; irreducible polynomials of a given degree playing the role of prime numbers of a given number of bits, we sometimes keep the "primes" terminology for convenience. Likewise, the discussion below uses terminology which is heavily borrowed from NFS implementation folklore. In particular the I and J parameters directly relate to the dimensions of the sieved area in what is customarily called the (i,j)-*plane* in the lattice sieving context.

For the two parameter sets we considered, we used a skewness of 2 (the skewness being the degree gap in the coefficients a and b of the functions $a(t)-xb(t)$ considered in the algorithm), and we put the special-q on the rational side, since we are in a range where the rational side yields larger norms than the algebraic side. We used a factor base bound of degree 23 (inclusive). The main difference between our two sets of parameters is the large prime bound.

Case 1: large prime bound of 27. In that case, we used $I = J = 15$, which is set to have enough relations per special-q, despite a low large prime bound. The threshold for deciding which candidates are passed to the cofactorization step after sieving is set to degree 81 for both sides, that is we allow three large primes of maximum degree on each side.

All the special-q of degree from 24 to 27 (inclusive) were sieved, producing a bit more than 52 million of relations (possibly non-unique). The relevant data is summarized in the following table. The running times are given for a single core of an Intel Core i5-2500 CPU running at 3.3 GHz. In particular, this assumes the presence of the PCLMULQDQ instruction. In practice, most of our computations were done using the idle time of a cluster[1] whose 4-year old processors do not support this instruction, and therefore run about twice slower.

$\deg q$	number of rels	s/rel	rels/sq	acc. rels	acc. time
24	6,940,249	1.48	9.93	6,940,249	2853 h
25	9,926,294	1.91	7.39	16,866,543	8119 h
26	14,516,775	2.42	5.62	31,383,318	17877 h
27	20,645,456	3.38	4.15	52,028,774	37260 h

Case 2: large prime bound of 28. In that case, we used $I = J = 14$, which was enough to get a decent rate of relations per special-q. The threshold was again set to 3 times the large prime bound, that is 84 for both sides. We sieved all the special-q from degree 24 to 28 (inclusive), and produced more than 117 million of relations, split as in the following table.

$\deg q$	number of rels	s/rel	rels/sq	acc. rels	acc. time
24	9,515,069	0.41	13.61	9,515,069	1083 h
25	13,816,908	0.54	10.29	23,331,977	3155 h
26	20,538,387	0.65	7.95	43,870,364	6863 h
27	29,652,781	0.86	5.96	73,523,145	13946 h
28	43,875,232	1.07	4.57	117,398,377	26986 h

In both cases, we obtained a number of relations that provided a reasonable excess.

1.3 Filtering

The filtering step is split in 3 stages:

- duplicate: remove duplicate relations from the relation collection step;
- purge: remove singletons (ideals that appear in only one relation) and remove relations while the excess is positive (*i.e.*, there is still more relations that ideals);
- merge: beginning of Gaussian elimination.

The filtering step was performed using the implementation described in [4]. It was run on the two sets of relations produced by the relation collection step. The same parameters were used in both cases.

Case 1: large prime bound of 27. In total $52,028,774$ relations were collected. They produced $30,142,422$ unique relations (42% duplicates). After the first singleton removal, about 29M relations remained as well as 19M ideals (so the excess was around 10M). At the end of the purge algorithm, there were 9.6M relations and as many ideals. The final matrix (after the merge algorithm) had 3.68M rows and columns (with, in average, 100 non-zero coefficients per row, which is close to optimal for our linear algebra implementation).

[1]We acknowledge the support of the Région Lorraine and the CPER MISN TALC project who gave us access to this cluster.

Case 2: large prime bound of 28. In total $117,398,377$ relations were collected. They produced $67,411,816$ unique relations (43% duplicates). After the first singleton removal, about 65M relations remained as well as 37M ideals (so the excess was around 28M). At the end of the purge algorithm, there were 13.6M relations and as many ideals. The final matrix (after the merge algorithm) had 4.85M rows and columns (with, in average, 100 non-zero coefficients per row).

For the actual computation, relations collected with both values of the large prime bound were considered to produce the matrix. This is of course not "fair", in the sense that if the computation were to be run again, we would have only one of the two relation sets. On the other hand it was a pity not to use all what we had at hand to reduce the cost of the linear algebra.

Starting from an input set of 78.8M unique relations, we obtained a matrix with 3,602,667 rows and columns, and 100 non-zero coefficients per row on average.

1.4 Linear algebra

The linear system to be solved is $Mw = 0$, where M is the matrix produced by the filtering step. We solved the linear system modulo the subgroup of order ℓ, which is a 202-bit prime.

The computation was carried out on NVIDIA GPUs. The implementation used a Residue Number System (RNS) arithmetic to accelerate modular operations over $\mathbb{Z}/\ell\mathbb{Z}$, since this representation system offers the opportunity to increase the parallelism between the computation units, and fits well with the GPU scenario. This approach is described in [6].

Two independent computations were completed, and the choice of these two setups was driven by the hardware which was available to us at the time of the computation.

- A simple Wiedemann algorithm was run on a single node equipped with two NVIDIA GeForce GTX 680 graphic processors. The computation time for this setup sums up to 18 days: 12 days on both GPUs for the initial sequence computation, 35 minutes for the minimal polynomial computation, and 6 days on both GPUs for the computation of the kernel element.

- Another option was tried, using a different computing facility[2] equipped with slightly different hardware. We used 4 distinct nodes, each equipped with two NVIDIA Tesla M2050 graphic processors, and ran the block Wiedemann algorithm with blocking parameters $m = 8$ and $n = 4$. The initial sequence computation required 2.6 days in parallel on the 4 nodes. The linear generator computation required 2 hours in parallel using 16 jobs on a 4-node cluster with Intel Core i5-2500 CPUs (3.3GHz) connected with Infiniband QDR network. Computation of the kernel vector required 1.8 days in parallel on the 4 GPU nodes.

1.5 Descent

Once the discrete logarithms of almost all elements up to the large prime bound have been found, we compute individual logarithms using the classical strategy of descent by special-q.

More precisely, we start by splitting the target element into the quotient of two elements of about half the degree, using an Euclidean algorithm that we stop in the middle. Randomizing the target allows to repeat that step until the two elements are smoother than average. In our case, after a dozen of minutes, we managed to rewrite the target in terms of elements of degree less than 90 (in comparison, straight out of the Euclidean algorithm, we have a numerator and a denominator whose degree is at most 405).

Then we "descended" these elements of degree less than 90 but above 28, by considering them as special-q in the relation collection software, so that they are rewritten as ideals of smaller degree. Hence a tree is built, where the discrete logarithm of a node can be deduced from the discrete

[2]This work was realized with the support of HPC@LR, a Center of Competence in High-Performance Computing from the Languedoc-Roussillon region, funded by the Languedoc-Roussillon region, the European Union and the Université Montpellier 2 Sciences et Techniques. The HPC@LR Center is equipped with an IBM hybrid Supercomputer.

logarithms of each of its children, which are of smaller degree. One of the degree 28 ideals involved in the tree was not known from the linear algebra step, and was therefore "re-descended" to other degree 28 ideals.

The overall cost of the individual logarithm step is less than one hour, and therefore was not thoroughly investigated.

2 Balancing sieving and linear algebra

In retrospect, it is now clear that the strategy of using a large prime bound of 27 is better than 28: in the same amount of sieving time, one obtains a post-merge matrix that is smaller.

The question of where to stop sieving is not so easy to answer in advance, but with the data that we have collected, we can give some hints for future choices.

With this objective in mind, we have run the filtering step for various numbers of relations (always produced with a large prime bound of 27), and estimated both the sieving time for getting these relations, and the linear algebra time for the corresponding matrix. The relations were added in increasing lexicographical order of the special-q. For the linear algebra cost, we used the quantity: size times total weight, which is theoretically proportional to the running time. With this arbitrary unit, the linear algebra step described in Section 1.4 has a cost of about 1298 and corresponds to 36 days on one GTX 680 GPU.

# rels	size after singleton	matrix size after merge	linalg cost (arbitrary unit)	sieve CPU time ($\times 10^3$ h)	linalg GPU time ($\times 10^3$ h)
27.7M	14.1M × 14.0M	4.99M	2493	15.4	1.65
31.3M	16.6M × 15.1M	4.46M	1995	17.8	1.32
33.9M	18.6M × 16.1M	4.28M	1837	20.2	1.22
36.5M	20.4M × 16.8M	4.15M	1723	22.7	1.14
39.1M	22.1M × 17.4M	4.04M	1633	25.1	1.08
41.7M	23.7M × 17.9M	3.94M	1560	27.5	1.03
44.2M	25.1M × 18.3M	3.87M	1498	29.9	0.99
46.8M	26.5M × 18.6M	3.80M	1444	32.4	0.96
49.4M	27.7M × 18.9M	3.73M	1396	34.8	0.92
52.0M	28.9M × 19.1M	3.68M	1354	37.2	0.90

In terms of cost per unit, or in term of electric power needed, one GTX 680 CPU card corresponds more or less to 10 cores of i5-2500. With this scaling, the second line of the table is optimal, with a running time equivalent to 31,000 hours on one core of i5-2500.

This conversion between GPU and CPU is admittedly disputable. For instance, in our case, we had only few GPU resources available compared to CPU, so the last line of the table was more suitable.

3 Summary, result

We took the element t as the basis of the discrete logarithm modulo

$$\ell = 41483867312606056475251865474888423964616257742413275679781 37.$$

Then, the logarithms of the elements of the factor base were readily available after the linear algebra step. For instance:

$$\log_t(t+1) \equiv 10708210517160253543158298743679898652591427309486848857025 74 \mod \ell.$$

As an illustration of the descent step, we computed the discrete logarithm of a "random" element. We decided to step away from the tradition of taking the logarithm of decimals of π,

and took as a "random" input the integer RSA-1024 (converted into a polynomial of degree 1023 in t using the same encoding as above). It was reduced modulo $\varphi(t)$ before taking its discrete logarithm. We get:

$$\log_t(\texttt{RSA1024}) \equiv 29997870719116434854500200834208349771549879083381254164707 96 \mod \ell.$$

References

[1] Shi Bai, Alain Filbois, Pierrick Gaudry, Alexander Kruppa, François Morain, Emmanuel Thomé, and Paul Zimmermann. CADO-NFS, Crible Algébrique: Distribution, Optimisation - Number Field Sieve. `http://cado-nfs.gforge.inria.fr/`.

[2] Razvan Barbulescu. Selecting polynomials for the Function Field Sieve. Preprint, 23 pages, available at `http://hal.inria.fr/hal-00798386`, 2013.

[3] Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau, and Paul Zimmermann. The relationship between some guy and cryptography. ECC2012 rump session talk (humoristic), see `http://ecc.2012.rump.cr.yp.to/`, 2012.

[4] Cyril Bouvier. The filtering step of discrete logarithm and integer factorization algorithms. Preprint, 20 pages, available at `http://hal.inria.fr/hal-00734654`, 2012.

[5] Jérémie Detrey, Pierrick Gaudry, and Marion Videau. Relation collection for the Function Field Sieve. In *Proceedings of ARITH-21*. IEEE, 2013. To appear, available at `http://hal.inria.fr/hal-00736123`.

[6] Hamza Jeljeli. Accelerating iterative SpMV for Discrete Logarithm Problem using GPUs. Preprint, 8 pages, available at `http://hal.inria.fr/hal-00734975`, 2012.

[7] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic. Preprint, 12 pages, available at `http://eprint.iacr.org/2013/095`, 2013.

[8] Antoine Joux and Reynald Lercier. Discrete logarithms in GF(2^{607}) and GF(2^{613}). E-mail to the NMBRTHRY mailing list; `http://listserv.nodak.edu/archives/nmbrthry.html`, September 2005.

Magma verification script

```
UP<t> := PolynomialRing(GF(2));
function hex2pol(x) return UP!Intseq(StringToInteger(x, 16), 2); end function;

phi := hex2pol("3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE8000000000000000000000CC0C\
FAEB0E0000000000000000000000000000004DFFFFFFFFFFFFFC3C3FFC3C3C3FCE3C2133FFFF\
FFFFFFFFE9697FE96804C84C97E0B0000C00CF9B0F7675354E79F4CF7C97E29");
assert Degree(phi) eq 809 and IsIrreducible(phi);

rsa1024 := hex2pol("C05748BBFB5ACD7E5A77DC03D9EC7D8BB957C1B95D9B206090D83FD1B6\
7433CE83EAD7376CCFD612C72901F4CE0A2E07E322D438EA4F34647555D62D04140E1084E999BB\
4CD5F947A76674009E2318549FD102C5F7596EDC332A0DDEE3A355186B9A046F0F96A279C1448A\
9151549DC663DA8A6E89CF8F511BAED6450DA2C1CB");
N := 2^809-1;
ell := Factorization(N)[1][1];
Log1024 := 29997870719116434854500200834208349771549879083381254164707 96;

Modexp(rsa1024, (N div ell), phi) eq Modexp(t, (N div ell)*Log1024, phi);
```