Computation of a 768-bit prime field discrete logarithm

Thorsten Kleinjung[1,2],
Claus Diem[2], Arjen K. Lenstra[1], Christine Priplata[2], and Colin Stahlke[2]

[1] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
[2] Universität Leipzig, Mathematisches Institut, D-04009 Leipzig, Germany

Abstract. This paper reports on the number field sieve computation of a 768-bit prime field discrete logarithm, describes the different parameter optimizations and resulting algorithmic changes compared to the factorization of a 768-bit RSA modulus, and briefly discusses the cryptologic relevance of the result.
Keywords: Discrete logarithm, DSA, ElGamal, number field sieve

1 Introduction

Let $p = [2^{766}\pi] + 62762$, which is the smallest 768-bit prime number larger than $2^{766}\pi$ for which $\frac{p-1}{2}$ is prime too*. Let $g = 11$, which is a generator of the multiplicative group $\mathbf{F}_p^\times$ of the prime field $\mathbf{F}_p$. On June 16, 2016, we finished the computation of the discrete logarithm of $t = [2^{766}e]$ with respect to g. We found that the smallest non-negative integer x for which $g^x \equiv t \bmod p$ equals

$$32592361791827056223861598597862370912834133883372105854395081352176815629509$$
$$16383480306379202371756381173524422992340416587484710799119774978643019959726$$
$$38266781162575370644813703762423329783129621567127479417280687495231463348812.$$

By itself, this is a useless result. What is interesting is how we found it, that we did so with much less effort than we expected, and what the result implies for cryptographic security that relies on the difficulty of larger similar problems. These issues are discussed in this paper.

The result was obtained using the number field sieve (NFS, [28,14]). It required the equivalent of about 5300 core years on a single core of a 2.2GHz Xeon E5-2660 processor, mostly harvested during the period May to December, 2015, on clusters at the authors' universities. On average each additional discrete logarithm requires two core days. This result is a record for computing prime field discrete logarithms. It closes the gap between record calculations for general purpose integer factoring and computing arbitrary prime field discrete logarithms, with the 768-bit integer factorization record [21] dating back to 2009. Although our effort was substantial, we spent a fraction of what we originally expected. The purpose of this paper is to describe how this was achieved.

Records of this sort are helpful to get an impression of the security offered by cryptographic systems that are used in practice. The 768-bit number field sieve factorization from [21], for instance, required about 1700 core years. Because factoring a single 1024-bit RSA modulus [34]

* Here $[x]$ denotes the classical entier function, the largest integer less than or equal to x.

using the number field sieve is about three orders of magnitude more work (cf. end of Section 2), an educated guess follows for the worst-case effort to break a 1024-bit RSA key. Interpretation of the resulting estimate is another matter. Depending on one's perception, applications, incentives, taste, ..., it may boost or undermine one's confidence in the security of 1024-bit RSA moduli.

The ratio is similar between the difficulties of computing 768-bit and 1024-bit prime field discrete logarithms (cf. Section 2). It follows that even the nonchalant users of 1024-bit RSA, ElGamal [11], or DSA [36] have no reason to be nervous anytime soon if their concern is an "academic attack" such as the one presented here (cf. [6]). They have to be a bit more concerned, however, than suggested by [2, Section 4.1]. Also, we explicitly illustrate in Section 3 that continued usage of 1024-bit prime field ElGamal or DSA keys is much riskier than it is for 1024-bit RSA (all are still commonly used), because once a successful attack has been conducted against a single well-chosen prime field all users of that prime field [27, Section 4] may be affected at little additional effort [2].

As shown in Section 5 our result gives a good indication for the difficulty of computing discrete logarithms in multiplicative groups of other 768-bit prime fields as well. One such group, the so-called First Oakley Default Group, is of some historical interest as it was one of the groups supported by the Internet Key Exchange standard from 1998 [16], a standard that has been obsolete since 2005 [17]. In some cryptographic applications, however, one may prefer to use a generator of a relatively small prime order subgroup of $\mathbf{F}_p^\times$ that is chosen in such a way that comparable efforts would be required by Pollard's rho in the subgroup and by the number field sieve in $\mathbf{F}_p^\times$. Our choice of p assures that no (published) shortcut can be taken for our discrete logarithm computation. It also represents the most difficult case for the number field sieve, in particular for its linear algebra step. It follows from the numbers presented below that, for a discrete logarithm computation, our choice is overall more difficult than a subgroup order that may sometimes be preferred for cryptographic applications. With independently optimized parameters the two efforts are however of the same order of magnitude (cf. Section 4).

Two simple methods can be used to give an a priori estimate of the effort to solve our 768-bit prime field discrete logarithm problem. The first is direct extrapolation (cf. Section 2): given that solving a 596-bit prime field discrete logarithm problem took 130 core years (cf. [7]), extrapolation suggests that our 768-bit problem should be doable in about thirty thousand core years. For the second method we observe that the number field sieve for factoring or for prime field discrete logarithms is essentially the same algorithm. When applied to 768-bit composites or 768-bit prime fields and when using comparable number fields, they deal with similar probabilities and numbers of comparable sizes, with the sole exception occurring in the linear algebra step: although in both cases the matrix is very sparse and all non-zero entries are (absolutely) very small, when factoring linear algebra is done in a matrix modulo two, but for discrete logarithm problems the matrix elements are taken modulo the group order (a 767-bit integer in our case). An opposite effect, however, is caused by the fact that, with proper care, the number fields will *not* be comparable because modulo large primes polynomial selection methods can be used that do not work modulo large composites.

It follows that the numbers reported in [21] can be used to derive an *upper bound* for the 768-bit prime field discrete logarithm effort, simply by using a 767-fold increase (cf. Section 3) of the linear algebra effort from [21] while leaving the other steps unchanged. With [21, Section 2.4] we find that fifty thousand core years should suffice for our problem. If we would switch to a

768-bit prime that allows a much smaller but cryptographically still interesting subgroup this rough overall estimate would be reduced by a factor of about five.

Thirty or fifty thousand core years would be a waste of resources for a calculation of this sort, and the more doable small subgroup alternative would be of insufficient interest; independent of our estimates, a very similar figure was derived in [2, Section 4.1]. All these estimates, however, overlook several points. Direct extrapolation of the 596-bit effort turned out to be meaningless due to software improvements and because the limited size did not allow an optimization that applies to our case. But more importantly, the very different nature and size of the moduli used in, respectively, the polynomial selection and linear algebra steps imply a radical shift in the trade-off between the steps of the number field sieve, which in turn leads to very different parameter and algorithmic choices compared to what is done for factoring. We are not aware of a satisfactory theoretical analysis of this different trade-off and the resulting parameter selection, or of a reliable way to predict the practical implication for the relative hardness of integer factoring and prime field discrete logarithm problems. It is clear, however, that the issue is more subtle than recognized in the literature, such as [31,26] and, more recently, [2, Section 4.1].

As described in Section 3, adapting the parameter choices and algorithms to the case at hand – and guided by multiple experiments – it was found that it should be possible to reduce the fifty thousand core years estimate by almost an order of magnitude. This led to the conclusion that actually doing the full calculation would be a worthwhile undertaking: in the first place because it shows that for our current range of interest k-bit factoring and computing k-bit prime field discrete logarithms require a comparable effort; and in the second place, and possibly more interesting, because it required more than just a casual application of known methods.

The previous 596-bit and current 768-bit prime field discrete logarithm records should not be confused with extension field discrete logarithm records. Due to recent developments, we now have much better methods than the number field sieve to compute discrete logarithms in small characteristic extension fields. As a consequence, those fields are no longer relevant for basic cryptographic applications such as DSA. Indeed, recent extension field records imply that impractically large extension fields would have to be used to get an appreciable level of security: for instance, computing discrete logarithms in the multiplicative group of the 9234-bit field $\mathbf{F}_{2^{2\cdot 3^5\cdot 19}}$ took less than fifty core years [15], and in the 3796-bit group $\mathbf{F}^{\times}_{3^{5\cdot 479}}$ the problem was dealt with in less than a single core year [19]. On the other hand, the current characteristic two *prime extension degree* record involved the much smaller finite field $\mathbf{F}_{2^{1279}}$ and took between three and four core years [20]: the advantage of the new methods over the number field sieve strongly depends on properties of the extension degree, but for favorable degrees the advantage is much bigger than the advantage for the number field sieve when factoring special numbers (such as Mersenne or Fermat numbers) compared to general ones (such as RSA moduli).

While the correctness of the outcome of our calculation can simply be verified, independent validation of the other claims made in this paper requires access to suitable source code and data. We have established a long-standing tradition of open collaborations [30] with other leading researchers in this field (see [21,22] and the references therein) which applies to anything relevant for the present project as well.

The paper is organized as follows. Section 2 presents the background for the rest of the paper. Section 3 describes the impact of the parameter selection on the way one of the main steps of the number field sieve is best implemented for the problem solved here and lists all relevant details of our new record calculation. Section 4 gives more details about the trade-off between the main steps of the number field sieve, and presents estimates for the effort required to solve a discrete logarithm problem in a small subgroup. In Section 5 it is shown that our choice of $p = [2^{766}\pi] + 62762$ is not more or less favorable than other primes of the same size.

2 Algorithm overview

Descriptions of the number field sieve are available in the literature, ranging from the high level narrative [33] to the somewhat simplified and fully detailed versions in [29] and [28], respectively.

Index calculus method [24,25,1]. Let $\mathbf{F}_p$ be a finite field of cardinality p, identified with $\{0, 1, \ldots, p-1\}$ in the usual manner, and let g generate its multiplicative group $\mathbf{F}_p^\times$. To compute discrete logarithms with respect to g, an index calculus method fixes a so-called *factor base* $B \subset \mathbf{F}_p^\times$, collects more than $\#B$ multiplicative *relations* between the elements of $B \cup \{g\}$, and uses linear algebra modulo the order of g to determine for all elements of B their discrete logarithm with respect to g. Given this information, the discrete logarithm of any $h \in \mathbf{F}_p^\times$ is then found by finding a multiplicative relationship between h and the elements of $B \cup \{g\}$.

Doing more or less the same modulo a composite N (as opposed to modulo p) and using linear algebra modulo two (as opposed to modulo the order of g) an integer solution to $x^2 \equiv y^2 \bmod N$ may be found, and thus a chance to factor N by computing $\gcd(N, x - y)$. This explains the similarity in the algorithms for factoring and computing discrete logarithms as well as the difference between the matrices for factoring and discrete logarithms that was pointed out in the introduction. The effect of the prime p versus the composite N, as also mentioned in the introduction, is touched upon below and in Section 3.

Different index calculus methods vary mostly in the way the multiplicative relations are found. This affects the way B is chosen. For prime p for instance, relations may be collected by considering g^e for random integers e and keeping those that factor over B. With B the set of primes up to some bound b one would thus be collecting *b-smooth* g^e-values. Faster methods increase the smoothness probabilities by generating smaller values in $\{1, 2, \ldots, p-1\}$; select the values in an arithmetic progression so that *sieving* can be used to faster recognize smooth values; allow in relations a few *large primes* between b and a large prime bound b_ℓ; or they manage to combine those speedups. Dan Gordon [14] was the first to show how for prime p the ideas from the number field sieve for integer factorization [28] can be included as well. Many other variants have been proposed since then; the most accurate reference for the one used here is [35].

Relations in the number field sieve. A property of the number field sieve that sets it apart from the earlier index calculus methods is that for a relation two distinct numbers must be smooth (with both numbers asymptotically significantly smaller than the values considered before). Let f and g be two coprime irreducible polynomials in $\mathbf{Z}[X]$ of degrees d_f and d_g, respectively, chosen in such a way that they have a root m in common modulo p (see Section 3

for how this may be done). A relation corresponds to a coprime pair of integers (a, b) with $b \geq 0$ such that the two integers $\mathcal{N}_f(a, b) = b^{d_f} f(\frac{a}{b})$ and $\mathcal{N}_g(a, b) = b^{d_g} g(\frac{a}{b})$ are smooth with respect to appropriately chosen bounds.

This is, very briefly, explained as follows. The integer $\mathcal{N}_f(a, b)$ is essentially (except for the leading coefficient of f) the norm of $a - \alpha_f b \in \mathbf{Z}[\alpha_f] \subset \mathbf{Q}(\alpha_f)$, where α_f denotes a zero of f and $\mathbf{Q}(\alpha_f)$ is the algebraic number field $\mathbf{Q}[X]/(f(X))$. The smoothness of $\mathcal{N}_f(a, b)$ then implies a factorization into small prime ideals in $\mathbf{Q}(\alpha_f)$ of the ideal $(a - \alpha_f b)$ (cf. [8]). Noting that mapping α_f to the common root m results in a ring homomorphism φ_f from $\mathbf{Z}[\alpha_f]$ to $\mathbf{F}_p$, and defining α_g and φ_g in a similar manner for g, a relation (a, b) thus corresponds to factorizations of the ideals $(a - \alpha_f b)$ and $(a - \alpha_g b)$ that map, via φ_f and φ_g, respectively, to the same element $a - bm \in \mathbf{F}_p$.

The 768-bit factorization from [21] used degrees $d_f = 6$ and $d_g = 1$; consequently, the labels *algebraic* or *rational* were used to distinguish values and computations related to f or g, respectively. These intuitive labels can no longer be used here, because the primality of our modulus (p) offers flexibility in the polynomial selection that is not available for composite moduli and that resulted, as could be expected, in the "better" choices $d_f = 3$ and $d_g = 4$ for the present paper. Though the f- and g-related parts here are thus both algebraic, it will be seen that the g-part is easier to deal with, and thus, to some extent, corresponds to the rational side in [21]. Because f and g have rather different properties, different considerations come into play when selecting the factor bases for $\mathcal{N}_f(a, b)$ and $\mathcal{N}_g(a, b)$. The single factor base B is therefore replaced by two distinct factor bases, denoted by B_f and B_g. For the present purposes it may be assumed that B_f and B_g consist of the primes bounded by b_f and b_g. We make no distinction between f and g for the large prime bound (which is thus still denoted by b_ℓ, with $\#B_\ell$ denoting the number of primes bounded by b_ℓ), but may allow different numbers of large primes in $\mathcal{N}_f(a, b)$ and $\mathcal{N}_g(a, b)$, denoted by n_f and n_g.

Finding relations in the number field sieve. As each relation requires two numbers being smooth, collecting relations is a two-stage process: in the first stage pairs (a, b) for which $\mathcal{N}_f(a, b)$ is smooth are located; in the second stage, from the pairs found those for which $\mathcal{N}_g(a, b)$ is smooth as well are selected. Thus, the first stage treats the numbers that are least likely to be smooth, thereby minimizing the number of pairs to be considered for the second stage: switching the roles of f and g would have led to more pairs to be treated in the second stage. Depending on the factor base sizes, various methods may be used to find relations.

The search for relations is typically limited to a (large) rectangular region S of the lattice $\mathbf{Z}^2$. For the first stage (and numbers in the current range of interest) index-q sublattices L_q of $\mathbf{Z}^2$ are identified such that q divides $\mathcal{N}_f(a, b)$ for all pairs $(a, b) \in L_q$ and for primes q close to and often somewhat larger than b_f (these primes are referred to as *special q primes*). The process described below is repeated for different special q primes until, after removal of unavoidable duplicates, enough relations have been collected.

Given a special q prime, *lattice sieving* is conducted over a rectangular region S_q (which roughly approximates $L_q \cap S$), to locate (a, b) pairs for which $\mathcal{N}_f(a, b)$ is b_f-smooth (except for q and at most n_f large primes $\leq b_\ell$). The number of "surviving" pairs thus found is denoted by y_f. If y_f is large (and the pairs are not spread too widely as for instance in [9,22]), it is best to again use lattice sieving in S_q to collect from those y_f pairs the y_g pairs that are actually relations, i.e., for which $\mathcal{N}_g(a, b)$ is b_g-smooth as well (again with at most n_g large primes

$\leq b_\ell$). This is the regular approach to the second stage, and was used in [21]. But there are circumstances where the second stage is best done in another manner: in [22], for instance, factorization trees (cf. [12, Section 4] and [4]) were used. This is also the approach taken here, as further described in Section 3.

Effort required by the number field sieve. Let

$$E(x,c) = e^{\left(\left(\frac{64}{9}\right)^{\frac{1}{3}}+c\right)(\log x)^{\frac{1}{3}}(\log\log x)^{\frac{2}{3}}}$$

where logarithms are natural; this slight variation on a well-known and more common notation allows us to focus on what is of greatest interest in the present context. The current best heuristic expected effort to compute a discrete logarithm in $\mathbf{F}_p^\times$ using the number field sieve is $E(p, o(1))$, asymptotically for $p \to \infty$ [14]. This is the same as the effort $E(N, o(1))$ (for $N \to \infty$) to factor a composite N using the number field sieve [8]. This "same" should, however, be taken with a grain of salt because the $o(1)$ hides different functions for the two cases.

Optimal factor base sizes. The smoothness probabilities, the number of relations to be collected, and the dimension of the matrix handled by the linear algebra all increase with the smoothness parameters b_f, b_g, b_ℓ, n_f and n_g. The resulting trade-off leads to optimal factor base sizes $\#B_f$ and $\#B_g$, namely $E(p, o(1))^{\frac{1}{2}}$ for discrete logarithms and $E(N, o(1))^{\frac{1}{2}}$ for factoring. As noted above, even if $[\log p] = [\log N]$, in a given model of computation the optimal values for both factoring and discrete logarithm computation may be very different because the two $o(1)$-functions behave quite differently. Moreover, in practice the situation is further complicated because of the software and hardware actually used. Thus, naively using factor base sizes that worked well for a factoring problem for a similarly sized prime field discrete logarithm problem, as done in the introduction and despite the "correction" attempted there, will at best result in a rough upper bound. Section 3 discusses this issue in more detail.

Remark on using $E(x,c)$ in practice. The uncertain function hiding in the $o(1)$ makes it challenging to use $E(p, o(1))$ to give an absolute estimate for the effort to solve a discrete logarithm problem in $\mathbf{F}_p^\times$. It turns out, however, that a somewhat pessimistic indication can be obtained for the relative effort for $\mathbf{F}_{\bar{p}}^\times$ compared to $\mathbf{F}_p^\times$, for $\bar{p}$ not much bigger than p (say, $\bar{p} \leq p^{\frac{4}{3}}$), by dropping the $o(1)$. Obviously, this assumes similar software that suffers no ill side-effects nor profits from new optimizations when moving to the larger $\bar{p}$. The same works for factoring.

As an example, the three orders of magnitude difference between the efforts of factoring 768-bit and 1024-bit moduli, as mentioned in the introduction, follows from $\frac{E(2^{1024},0)}{E(2^{768},0)} \approx 1200$; the jump from 130 core years for a 596-bit prime field discrete logarithm problem to about thirty thousand core years for 768 bits follows from $\frac{E(2^{768},0)}{E(2^{596},0)} \approx 275$ – an extrapolation that failed to be useful because of the reasons mentioned in the introduction.

3 Computational details

This section provides some background on our parameter choices. For comparison, we also provide the parameters that were used for the 768-bit factoring effort from [21].

Polynomial selection. To get an initial impression of the feasibility of the calculation an extensive search was conducted using the method from [18]. First all integer polynomials g of degree

four with coefficients absolutely bounded by 165 (and noting that $g(X)$, $g(-X)$, and $X^4 g(\frac{1}{X})$ and thus $X^4 g(\frac{-1}{X})$ are equivalent) were inspected, by using, for all the roots of g modulo p, lattice reduction to find a corresponding degree three integer polynomial f, and measuring the overall quality of all resulting pairs (f, g) (as usual with respect to their small modular roots and size properties). For the second search, with bound 330, roots and size properties of g were first considered and only for the most promising candidates the roots of g modulo p were calculated and, if any, the polynomial f was derived. The best pair was found during the first search:

$$\begin{aligned} f(X) &= 37086340388641614115050552391952767723193261818410009592 4X^3 \\ &\quad - 19379813128330387785656174698293955440652559380159203096 79X^2 \\ &\quad - 21758329362694789978757744112833302761754109500473473641 5X \\ &\quad + 27726073040034952289042261847349814852870611500333793515 0, \\ g(X) &= 140X^4 + 34X^3 + 86X^2 + 5X - 55. \end{aligned}$$

Because it requires root finding modulo p, the above search does not work to find polynomials for the number field sieve for integer factorization. There one is limited to more restrictive methods that cannot be expected to result in polynomials of comparable "quality", with respect to the metric used in this context: indeed, the above pair is noticeably better than the degree (6,1) pair used for the slightly smaller 768-bit modulus factored in [21]. A more quantitative statement requires a more careful analysis than we are ready to provide here. No significant amount of time was spent on searching for pairs (f, g) of other degrees than $d_f = 3$ and $d_g = 4$.

Parameter selection background. The two main steps of the number field sieve after the polynomial selection, relation collection and linear algebra, are of a very different nature. Relation collection is long-term but low-maintenance: core years are easily harvested on any number of otherwise idle independent cores on any number of clusters that one can get access to, progress will be steady, and the process requires almost no human interaction. The results can easily be checked for correctness (cf. [22, Section 6]) and results that are lost or forgotten are easily replaced by others. Compared to this almost "happy-go-lucky" relation collection process, the linear algebra is tedious and cumbersome, despite the elegance of the block Wiedemann method used for it [37,10]. It involves careful orchestration of a (modest number of) substeps each of which requires as many tightly coupled cores as needed to store the data (easily on the order of hundreds of GB), frequent checkpointing, and a central step that is even more memory-demanding but otherwise fortunately relatively swift. Overall, based on past experience core years are collected at about half the rate compared to relation collection.

For both main steps the required effort is well understood:
• Given relation collection software and any choice of smoothness parameters a small number of experiments suffices to get an accurate indication for the effort required to collect any specified number of relations (it follows from the description in Section 5 how this may be done).
• Similarly, given block Wiedemann software and any matrix dimension, weight, and modulus-size, the overall linear algebra effort can be reliably estimated based on the effort required for a few matrix $\times$ vector multiplications on the processor network of one's choice.

However, the relations as collected are never directly used for the linear algebra, because doing so would be hugely inefficient. Instead, a linear algebra preprocessing step is applied to the relations in order to reduce the dimension of the matrix while keeping its weight under control, thereby (substantially) reducing the linear algebra effort. This preprocessing becomes

more effective as more relations are available (cf. Section 4) but the precise behavior of both dimension and weight depends on how (large) primes in a relation can be matched with the same primes in other relations and is thus uncertain. In practice one collects relations while occasionally doing a preprocessing attempt, and stops when the resulting linear algebra effort is within the targeted range. When to stop is a judgment call as more often than not the additional effort invested in relation collection is more than the expected linear algebra savings: it thus serves more to reduce the linear algebra headaches than to reduce the overall effort. As an example, for the current 768-bit factoring record about twice the strictly necessary relation collection effort was spent to make the linear algebra more manageable, an extra effort that was commented on as being "well spent" (cf. [21, Introduction]). These "negative returns" are further illustrated in Section 4.

Based on consistent past behavior of the preprocessing and given specific smoothness parameters, it can be roughly estimated how many relations have to be collected for a targeted matrix dimension and weight. Given the uncertainty alluded to above, this estimate can only be a guess, though it is a mildly educated one. With the known behavior of the software, an overall effort estimate assuming those specific smoothness parameters can be derived. Repeating this for different smoothness parameters, the "best" – despite a lack of clear optimization criteria – overall effort then follows.

Parameter selection. Our starting point was that on current equipment the linear algebra effort for the 768-bit modulus factored in [21] would be about 75 core years. Given the similarity of the algorithms and sizes, and using the same smoothness parameters as in [21], the overall effort to solve our discrete logarithm problem can be estimated as $1500 + 767 \cdot 75 = 59025$ core years; due to the small entries of the matrix the linear algebra effort only depends linearly on the size of the group order. The fifty thousand core years estimate mentioned in the introduction then follows from the expected favorable comparison of polynomials found using the method from [18] compared to the method used in [21]; we refer to the papers involved for an explanation of this effect.

All that is clear at this point is that attempts to lower this estimate must focus on lowering the linear algebra effort; thus the smoothness parameters must be reduced, but by how much and what the overall effect is going to be is unclear. Because the block Wiedemann effort is roughly proportional to the product of the matrix dimension and weight, reducing the matrix dimension by a factor of c while keeping the same average row-weight, cuts the linear algebra effort by a factor of c^2. Thus, given any targeted linear algebra effort a reduction factor c for the dimension follows. Assuming that, for our problem, a thousand core years would be acceptable for the linear algebra effort, a dimension reduction factor of about 7.6 follows, because $\frac{767 \cdot 75}{7.6^2} \approx 1000$. Compared to the parameters used in [21], such a drastic reduction requires severely cutting the smoothness parameters and the number of special q primes one may use. This in turn entails a more than proportional increase in the search space and thus a substantial increase in the overall relation collection effort compared to the 1500 core years spent in [21]. A priori, however, and as argued above, the effect of any of the changes that one may wish to consider cannot be accurately predicted.

While conducting experiments with a 640-bit example to better understand the increase in the relation collection effort depending on various possible combinations of smaller smoothness parameters and larger search spaces, we observed a mildly beneficial side-effect which – once observed – is obvious, but which was unanticipated: in the notation of Section 2, if B_f

decreases, the number y_f of b_f-smooth norms $\mathcal{N}_f(a,b)$ becomes smaller too, at a given point to an extent that it becomes more efficient to replace sieving for the second search stage (as used in [21]) by factorization trees. For the 640-bit example the effect was still small, i.e., y_f was still relatively large. But for our 768-bit prime the impact soon turned out to be considerable, almost halving the (inflated, compared to [21]) relation collection effort.

The resulting "best" parameters that we settled for are listed in Table 1 (though for some special q primes larger factor bases were used), along with the parameters used in [21] for the 768-bit number field sieve factorization. The clear difference is that the choices for 768-bit factoring were optimized for speed during relation collection (collecting relations until the after-preprocessing matrix dimension and weight were found to be acceptable), whereas our choices try to squeeze as many relations as possible out of every special q prime under a relatively restrictive smoothness regime. Compared to [21], $\#B_f$ is reduced by a factor of a bit more than two, the number of special q primes is reduced by a factor of more than twenty, the number of large primes per relation is cut from $4+3$ to $2+2$ with a large prime bound that is reduced by a factor of 2^4 from 2^{40} to 2^{36}, while $\#B_g$ remains unchanged and the search space is on average (forced to be) more than 2^8 times larger. As a result the number of relations per core unit of time drops by a factor of about sixteen compared to [21].

The first preprocessing attempt that resulted in the hoped-for matrix dimension and weight occurred when 1.09e10 relations had been collected, i.e., about six times fewer relations than in [21]. The resulting overall relation collection effort thus became $\frac{16}{6} \cdot 1500 = 4000$ core years. With 920 core years the linear algebra effort was close to but less than the thousand core years that we had hoped to achieve. A much smaller set of relations would in principle have sufficed too, but it would have resulted in a larger linear algebra effort; Section 4 below describes the trade-off in more detail.

Some of the details in Table 1 are listed for completeness; for explanations we refer to [21]. Like most of our computational number theory colleagues, we missed the fact (which apparently had not escaped numerical analysts) that the evaluation stage of the block Wiedemann method can be sped up considerably using Horner's rule [13]; it would have reduced our overall effort to approximately 5000 core years.

Database. The linear algebra step resulted in the (virtual) logarithms of 24 million prime ideals. Spending less than 200 core years for additional sieving and further processing, a database was built containing the about 3e9 logarithms of all prime ideals of norms up to 2^{35}.

Individual logarithms. Using the database and q-descent, the logarithm of the target t from the introduction was computed in approximately 115 core hours. Similar computations for $t+1, t+2, \ldots, t+10$ took on average about 96 core hours per logarithm.

With improved software any individual logarithm can now be computed at an average effort of 43 core hours (and a rather large variation, cf. Table 1). Further software enhancements are easily conceivable, but this already insignificant effort underscores the point made in the introduction that once a single number field sieve based attack has been carried out successfully, other attacks against the same prime field are straightforward.

Table 1: Comparison of 768-bit factoring and computing 768-bit prime field discrete logarithms.

	768-bit factorization from [21]	768-bit discrete logarithm
polynomial selection	2005 and 2007	2015:02:14 – 2015:05:04
	more than 2e18 pairs (f, g) were considered, spending 40 core years (20 in 2005 and 20 in 2007)	$\lvert g_{\max}\rvert \le 165$: all 5.9e11 g-candidates, 110 core years $\lvert g_{\max}\rvert \le 333$: best 2e13 g-candidates, 90 core years
$d_f,\ d_g$	6, 1	3, 4
relation collection	2007:08 – 2009:04	2015:05:04 – 2015:12:13
method	lattice sieving for both f and g	lattice sieving for f ($> 98\%$ of effort) factorization tree for g ($< 2\%$ of effort)
smoothness bounds	$\ge$ 2GB RAM: $b_f = \min(q, 1.1\text{e}9)$, $b_g = 2\text{e}8$, $\#B_f = 5.6\text{e}7$, $\#B_g = 1.1\text{e}7$ $<$ 2GB RAM: $b_f = 4.5\text{e}8$, $b_g = 1\text{e}8$, $\#B_f = 2.4\text{e}7$, $\#B_g = 5.8\text{e}6$	$b_f = \min(q, 4.4\text{e}8)$, $b_g = 2\text{e}8$, $\#B_f = 2.3\text{e}7$, $\#B_g = 1.1\text{e}7$
large primes parameters	$b_\ell = 2^{40}(\#B_\ell = 4.1\text{e}10), n_f = 4, n_g = 3$	$b_\ell = 2^{36}(\#B_\ell = 2.9\text{e}9), n_f = 2, n_g = 2$
$\#S_q$: bounds on q	2^{31}: $4.5\text{e}8 < q < 1.1\text{e}10$	2^{40}: $1.9\text{e}8 < q < 3\text{e}8$
$\#q$	4.8e8	5.7e6
time per q	< 100 seconds	≈ 8750 seconds
(y_f, y_g) per q	$((y_f$ large and irrelevant$), 134)$	$(7.9\text{e}5, 590)$
$\#S_q$: bounds on q		2^{39}: $3\text{e}8 < q < 6\text{e}8$
$\#q$		1.5e7
time per q		≈ 4950 seconds
(y_f, y_g) per q		$(7.3\text{e}5, 480)$
$\#S_q$: bounds on q		2^{38}: $6\text{e}8 < q < 6.3\text{e}8$
$\#q$		1.5e6
time per q		≈ 2550 seconds
(y_f, y_g) per q		$(4.6\text{e}5, 300)$
totals:		
number of special q primes	4.8e8	2.2e7
yield: with duplicates & unfactored	64 334 489 730	10 802 334 123†
yield: unique & factored	47 705 019 942	9 060 739 382
yield: free relations	57 223 462	19 967 617
effort	$\approx$ 1500 core years	$\approx$ 4000 core years
linear algebra preprocessing		
duplicate & singleton removal	2009:05	2015:08:11 – 2015:12:20
filtering	2009:06	2015:12:21 – 2015:12:25
result: dimensions	192 796 550 × 192 795 550	23 504 483 × 23 504 413
result: weight	27 797 115 920 "one" bits	3 140 911 353 entries of 767 bits
result: average non-zeros per row	144	134
effort	a few core years	a few core years
linear algebra		
block Wiedemann parameters	$m = 16 \times 64, n = 8 \times 64$	$m = 32, n = 16$
scalar products	2009:08:10 – 2009:11:03 8 independent sequences 43 core years (current cluster)	2015:12:28 – 2016:03:23 16 independent sequences 560 core years
Berlekamp-Massey	2009:11:03 17.3 hours on 224 of 672 cores 896GB RAM 0.5(+1 idle) core years (old cluster)	2016:04:03 – 2016:04:04 33.85 hours on 256 of 4096 cores 8TB RAM 1(+15 idle) core years
evaluation	2009:11:05 – 2009:12:09 many independent jobs 30 core years (current cluster)	2016:04:05 – 2016:05:18 480 independent jobs about 355 core years
effort	75 core years (current cluster)	920 core years
final calculation	2009:12:07 – 2009:12:12 20 core hours per square root attempt	2016:05:18 – 2016:06:16 200 core years to build database; 43 core hours on average (was 100) per individual logarithm, varying between 3 and 220 core hours

†: this excludes about 1e8 forgotten relations

4 Trade-off

During relation collection occasional preprocessing attempts were made, until the resulting matrix was found to be acceptable. Data about the non-final attempts were not kept, so for the purpose of the present paper part of this work was redone to be able to give an impression how the linear algebra effort decreases as more relations become available.

Table 2 summarizes the results of these "after the fact" preprocessing attempts of the sets of relations found for special q primes up to increasing bounds b_q, along with the resulting extrapolations of the block Wiedemann efforts (*not* using Horner's rule for the evaluation stage). Estimates are also listed for the effort required for a discrete logarithm problem in a 160-bit subgroup of the multiplicative group of a 768-bit prime field. For each set of relations up to five preprocessing attempts were made, and the best was selected depending on the subgroup size; this explains why for five of the 160-bit subgroup entries the matrix dimension and weight are different from those in the 767-bit subgroup entry. The last two rows show the effect of the 1e8 forgotten relations (cf. Table 1): including those and spending more time on constructing the matrix could have reduced the matrix effort by 20 (or 5) core years.

Table 2: Relation collection effort, matrix dimension and weight as a result of preprocessing, estimated linear algebra effort, and the combined effort (all efforts are in core years), when using special q primes up to b_q and both for 767-bit and 160-bit subgroup orders. The overshoot factor is the ratio of the number of relations and the number of relations for the least b_q (2.8e8) for which enough relations had been found. Relations are unique and factored and include the free relations.

	relation collection			767-bit subgroup order (our problem)					160-bit subgroup order				
b_q	relation count	effort	overshoot factor	dimension	weight	nodes	matrix effort	combined effort	dimension	weight	nodes	matrix effort	combined effort
2.70e8	2.33e9	insufficient											
2.80e8	2.58e9	1300	1.000	5.62e7	9.5e9	25	6575	7875	5.62e7	9.5e9	9	1780	3080
3.06e8	3.24e9	1625	1.255	3.27e7	6.2e9	12	2095	3720	4.00e7	4.7e9	4	500	2125
3.35e8	3.90e9	1850	1.508	2.96e7	4.5e9	9	1420	3270	2.96e7	4.5e9	4	325	2175
3.67e8	4.52e9	2100	1.751	2.62e7	4.4e9	9	1120	3220	2.76e7	3.9e9	4	270	2370
4.03e8	5.15e9	2400	1.995	2.47e7	4.2e9	9	1000	3400	2.57e7	3.8e9	4	240	2640
4.75e8	6.50e9	2975	2.516	2.36e7	3.7e9	9	870	3845	2.48e7	3.3e9	4	210	3185
5.37e8	7.74e9	3475	2.997	2.41e7	3.1e9	6	790	4265	2.41e7	3.1e9	4	190	3665
6.30e8	9.15e9	4000	3.542	2.17e7	3.6e9	6	740	4740	2.08e7	4.0e9	4	180	4180
(used)	9.08e9	4000	3.515	2.35e7	3.1e9	6	760	4760	2.35e7	3.1e9	4	185	4185

The difference between the linear algebra estimate in the last row for the matrix as actually used and the effort listed in Table 1 is due to a lower number of nodes on which the experiment was run: for a full execution it would lower the linear algebra effort, but increase the calendar time. The effort required for polynomial selection and individual logarithms is independent of the b_q-value, and is not included in the "combined effort". The database building effort may be up to three times larger for the smallest feasible b_q-value, but is not included either.

The numbers in the "combined effort" columns of Table 2 illustrate the negative returns mentioned in Section 3: with more patience to deal with a larger linear algebra problem (that would have required disproportionally more calendar time), our overall effort could have been reduced from 5300 to less than 4000 core years. As in [21], the additional relation collection effort was well spent, because a large block Wiedemann job requires constant attention and any way to reduce the calendar time is welcome.

Note that for the smaller subgroup problem the overall least effort is reduced by a factor smaller than two.

5 Other prime fields

To convince ourselves that our results were not due to unexpected, lucky properties of our choice of prime field, we tested ten other similarly chosen 768-bit primes and roughly compared them to our p with respect to their sieving yield. Define the following eleven transcendental or supposed-to-be transcendental numbers:

$\rho_0 = \pi$;
$\rho_1 = e$, Euler's number;
$\rho_2 = \gamma$, the Euler-Mascheroni constant;
$\rho_3 = \sqrt{2}^{\sqrt{2}}$;
$\rho_4 = \zeta(3)$, where ζ is the Riemann zeta function;
$\rho_5 = \log(\frac{1+\sqrt{5}}{2})$, the regulator of the "smallest" real quadratic number field;
$\rho_6 = \Omega_{X_0(11)}$, the real period of the "smallest" elliptic curve, namely $X_0(11)$ given by $y^2 + y = x^3 - x^2 - 10x - 20$;
$\rho_7 = \hat{h}_{X_0(37)}(P_{37})$, the canonical height of a generator $P_{37} = (0, 0)$ of the "smallest" elliptic curve of rank 1, namely $X_0(37)$ given by $y^2 + y = x^3 - x$;
$\rho_8 = t_0$, the imaginary part of the first zero $\frac{1}{2} + t_0 i$ on the critical strip of ζ;
$\rho_9 = \pi^e$;
$\rho_{10} = \sum_{i=1}^{\infty} 10^{-i!}$, Liouville's constant.

For $0 \le i \le 10$ let $\epsilon_i = 767 - [\frac{\log \rho_i}{\log 2}]$ and let p_i be the least prime larger than $2^{\epsilon_i}\rho_i$ for which $\frac{p_i-1}{2}$ is prime as well. Then $p_0 = p$. Let π_j be the number of primes in $[j \cdot 1e7, (j+1) \cdot 1e7]$; for $19 \le j \le 62$ these intervals cover our range of special q primes (cf. Table 1).

For each of the eleven primes p_i with $0 \le i \le 10$ the following calculation was carried out:
Polynomial selection. Find the best pair (f_i, g_i) for p_i among the first 5e9 candidate polynomials for g_i. (This requires about one core year.)
Sieving experiments. For $19 \le j \le 62$ find the number r_j of relations when sieving with the parameters as in Table 1 but with the polynomials f_i and g_i and the prime p_i and for the least special q prime larger than $j \cdot 1e7 + 5e6$. (This requires less than four core days, cf. Table 1.)
Overall yield estimate. Let $R_i = \sum_{j=19}^{62} \pi_j r_j$.

Table 3: Relative performance of $p = p_0$ compared to ten other choices.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
p_i/p_0	1.000	0.865	0.735	1.039	0.765	1.225	0.808	1.041	1.125	0.894	1.120
R_i/R	0.844	0.821	0.847	0.820	0.864	0.800	0.795	0.884	0.848	0.798	0.823
R_i/R_0	1.000	0.973	1.004	0.972	1.024	0.948	0.942	1.048	1.005	0.946	0.975

We also carried out the same sieving experiments for the polynomial pair (f, g) from Section 3 and $p = p_0$, finding an overall yield estimate $R = 1.02e10$. This is less than the 1.09e10 relations reported in Table 1, because there some of the sieving jobs used a larger factor base bound than reported in Table 1, thus producing more duplicates. But it is more than R_0 (which was found to be 8.6e9), matching the expectation that (f, g) is considerably better than (f_0, g_0). Table 3 lists the relative performance of our p compared to the ten new choices: as can be seen in the final row, four of the ten perform better and six are worse, but they are all within a 6% margin from p. It also follows that the core years spent on polynomial selection for our p were well spent.

Although our tests counter suspicions about p being special, it may be argued that in practice primes used in cryptography would be chosen with high entropy [5]. Testing a few "random" primes as well might strengthen our argument. It is unclear to us, however, how such primes may be obtained in a manner that is sufficiently convincing to any suspicious reader, without input from that reader [32].

6 Conclusion

We presented the computation of a discrete logarithm in the multiplicative group of a 768-bit prime field. This is a new record in its category, beating the previous 596-bit record. We showed the beneficial effect of judicious choice of parameters and algorithms, and highlighted the differences with integer factorization. Based on our findings we may conclude that for sizes that are currently within reach of an academic effort, the hardness of factoring and computing prime field discrete logarithms is comparable, though discrete logarithms are harder. Although this was always suspected to be the case, the gap between the two problems is quite a bit smaller than we expected. Compared to the 768-bit factoring record (which required 1700 core years as opposed to our 5300 core years) we used less calendar time and a smaller collaborative and less heterogeneous effort [23]. We also conclude that the explicit 1024-bit estimates from [2, Section 4.1] should be redone, as they require not entirely straightforward re-optimization efforts.

Unless algorithmic improvements are proposed or new insights may be expected, pushing for actual new factoring or prime field discrete logarithm records – as opposed to studies that result in reliable estimates – is mostly a waste of energy. We are not aware of any developments based on which we could realistically expect publication of a 1024-bit record within the next, say, five years. As usual, this may change at any moment, but so far the predictions made back in 2009 (cf. [6]) have already turned out to be accurate, or remain valid. In this context it is relevant to note that the project embarked on in [3] is still ongoing.

Acknowledgements. We thank Rob Granger and the anonymous Eurocrypt 2017 reviewers for their useful comments. Part of the computation was carried out on equipment sponsored by the Swiss National Science Foundation under grant number 206021-144981.

References

1. L. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *FOCS*, pages 55–60, 1979.
2. D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.
3. D. V. Bailey, B. Baldwin, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, G. van Damme, G. de Meulenaer, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, C. Paar, F. Regazzoni, P. Schwabe, and L. Uhsadel. The Certicom challenges ECC2-X. Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS 2009, 2009. `http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf`.
4. D. J. Bernstein. How to find small factors of integers. `http://cr.yp.to/papers.html`, june 2002.
5. D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, T. Lange, R. Niederhagen, and C. van Vredendaal. How to manipulate curve standards: a white paper for the black hat. Cryptology ePrint Archive, Report 2014/571, 2014. `http://eprint.iacr.org/2014/571`.

6. J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. On the security of 1024-bit RSA and 160-bit elliptic curve cryptography. Cryptology ePrint Archive, Report 2009/389, 2009. `http://eprint.iacr.org/`.
7. C. Bouvier, P. Gaudry, L. Imbert, H. Jeljeli, and E. Thomé. Discrete logarithms in GF(p) – 180 digits. NMBRTHRY list, 11/6/2014.
8. J. P. Buhler, H. W. Lenstra Jr., and C. Pomerance. Factoring integers with the number field sieve. pages 50–94 in [28], 1992.
9. D. Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.
10. D. Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.
11. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. Blakley and D. Chaum, editors, *Crypto 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, Heidelberg, 1985.
12. 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 *Lecture Notes in Computer Science*, pages 194–207. Springer, Heidelberg, 2004.
13. P. Gaudry, September 2016. Private communication: A nice trick by Emmanuel Thomé.
14. D. M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6:124–138, 1993.
15. R. Granger, T. Kleinjung, and J. Zumbrägel. Discrete Logarithms in $GF(2^{9234})$. NMBRTHRY list, 31/1/2014.
16. IETF. RFC 2409. `https://tools.ietf.org/html/rfc2409`, November 1998.
17. IETF. RFC 4306. `https://tools.ietf.org/html/rfc4306`, December 2005.
18. A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Mathematics of Computation*, 72(242):953–967 (electronic), 2003.
19. A. Joux and C. Pierrot. *Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms*, pages 378–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
20. T. Kleinjung. Discrete logarithms in GF(2^{1279}). NMBRTHRY list, 17/10/2014.
21. 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 *Lecture Notes in Computer Science*, pages 333–350. Springer, Heidelberg, 2010.
22. T. Kleinjung, J. W. Bos, and A. K. Lenstra. Mersenne factorization factory. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 358–377, 2014.
23. T. Kleinjung, J. W. Bos, A. K. Lenstra, D. A. Osvik, K. Aoki, S. Contini, J. Franke, E. Thomé, P. Jermini, M. Thiémard, P. Leyland, P. L. Montgomery, A. Timofeev, and H. Stockinger. A heterogeneous computing environment to solve the 768-bit RSA challenge. *Cluster Computing*, (15):53–68, 2012.
24. M. Kraitchik. *Théorie des nombres, Tome I.* Gauthiers-Villars, Paris, 1922.
25. M. Kraitchik. *Recherches sur le théorie des nombres, Tome I.* Gauthiers-Villars, Paris, 1924.
26. A. K. Lenstra. Unbelievable security: Matching AES security using public key systems. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 67–86. Springer, Heidelberg, 2001.
27. A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064, 2012. `http://eprint.iacr.org/2012/064`.
28. A. K. Lenstra and H. W. Lenstra Jr. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.
29. A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.
30. A. K. Lenstra and M. S. Manasse. Factoring by electronic mail. In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT 1989*, volume 434 of *Lecture Notes in Computer Science*, pages 355–371. Springer, Heidelberg, 1990.
31. A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
32. A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `http://eprint.iacr.org/2015/366`, to appear in the International Journal of Applied Cryptology as *Trustworthy public randomness with sloth, unicorn, and trx.*

33. C. Pomerance. A tale of two sieves. *Notices of the AMS*, 43(12):1473–1485, December 1996.
34. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
35. O. Schirokauer. Virtual logarithms. *J. Algorithms*, 57(2):140–147, 2005.
36. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-4, 2013. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.
37. D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32:54–62, 1986.