The State of the Art in Integer Factoring and Breaking Public-Key Cryptography

Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger,
Emmanuel Thomé, Paul Zimmermann

The state of the art in integer factoring and breaking public key cryptography

Fabrice Boudot[1], Pierrick Gaudry[2], Aurore Guillevic[2], Nadia Heninger[3], Emmanuel Thomé[2], and Paul Zimmermann[2]

[1]Université de Limoges, XLIM, UMR 7252, F-87000 Limoges, France
[2]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[3]University of California, San Diego, USA

The security of essentially all public-key cryptography currently in common use today is based on the presumed computational hardness of three number-theoretic problems: integer factoring (required for the security of RSA encryption and digital signatures), discrete logarithms in finite fields (required for Diffie-Hellman key exchange and the DSA digital signature algorithm), and discrete logarithms over elliptic curves (required for elliptic curve Diffie-Hellman and ECDSA, Ed25519, and other elliptic curve digital signature algorithms).

In this column, we will review the current state of the art of cryptanalysis for these problems using classical (non-quantum) computers, including in particular our most recent computational records for integer factoring and prime field discrete logarithms.

1 Integer Factoring and RSA

Integer factorization is probably the most well known of the computational problems we discuss in this article.

The RSA cryptosystem is "based on" the hardness of integer factoring in that if an adversary can factor the public modulus N contained in an RSA public key into its prime factors p and q, they can easily compute the private key. Interestingly, although factoring the public modulus N appears to be the best approach to break RSA, these problems are not known to be equivalent: there might be an algorithm to efficiently decrypt RSA ciphertexts or forge digital signatures that does not require or reveal the factorization of the modulus.

Factoring small numbers via trial division is easy: decomposing 91 into 7 times 13, or 2701 into 37 times 73. Even finding factors of large numbers may also be easy: one can see that the integer 26674 is divisible by 2. But an algorithm that can efficiently factor any integer with hundreds of decimal digits is still not known.

The integer factorization problem is easy to state, and mathematicians had already developed factorization algorithms that ran much faster than brute force before the 1970s, when the problem of factorization became a cornerstone of public-key cryptography. There is a wide variety of classical algorithms that can factor integers of various forms: Pollard's *rho* algorithm can find a factor p of an integer in time $O(\sqrt{p})$, Pollard's $p-1$ algorithm can efficiently find a factor p of an integer if the value $p-1$ has only "small" prime factors, Fermat factorization can efficiently factor an integer $N = a^2 - b^2$ if a and b are "close" to $\sqrt{N}$, and so on. The construction of the RSA modulus N is designed to evade these "special-purpose" factoring algorithms. This leaves the cryptanalyst no choice but to use a so-called "general purpose" factoring algorithm, which we will consider next.

Currently, the most efficient algorithm to factor RSA moduli is called the General Number Field Sieve (GNFS, or sometimes just NFS). GNFS is described in [8], but its ancestry can be traced to a few years earlier as a means to factor integers of a particularly suitable form [8, 7].

1.1 Sketch of the NFS algorithm

Let $N = pq$ be a composite number. We will illustrate the algorithm by factoring the toy example $N = 2701$. Observe that modulo N, a number has up to four square roots; for example $(\pm 119)^2 = (\pm 1287)^2 = 656 \mod N$. A suitable pair of these square roots can be used to factor N: we have $119^2 - 1287^2 = (119+1287)(119-1287) = 0 \mod N$. If we are lucky, the factors of this difference of squares split the factors of N, and we can use the GCD algorithm to find these factors. In our example, we have $\gcd(119+1287, N) = 37$ and $\gcd(119-1287, N) = 73$. If we got unlucky, we can simply look for more congruences of the form $X^2 = Y^2 \mod N$ until we find one such that $X \neq \pm Y \bmod N$.

This basic idea dates back to Fermat factorization, but a linear search for congruences of squares does not lead to an efficient factoring algorithm. To improve the running time, one can instead *construct* squares as products of congruences of known factorization modulo N.

This results in a two-stage algorithm. First, in the *relation collection step*, one collects many multiplicative relations modulo N. The desired relations are *smooth*, meaning that each side factors completely into relatively small primes. Next, to construct a relation of squares, the algorithm observes that a square integer has even exponents in its prime factorization, and thus once enough relations have been collected, one can do *linear algebra* over the vector of exponents modulo 2 to solve the system and obtain a congruence of squares.

Returning to our toy example $N = 2701$, one has $52^2 = 3 \bmod N$ and $53^2 = 2^2 \cdot 3^3 \bmod N$, so that $(52 \cdot 53)^2 = 55^2 = 18^2 \bmod N$, and $\gcd(55 + 18, N) = 73$.

The Quadratic Sieve and CFRAC (Continued FRACtion method) factoring algorithms follow the general approach outlined above. The expected running time of these algorithms depends on optimizing the chosen *smoothness bound* for desired relations. A lower smoothness bound means that the algorithm does less work to test a candidate relation, and less work in the linear algebra phase, but one must iterate through more candidate relations in order to find suitably smooth relations.

The Number Field Sieve also uses the above approach, but it takes advantage of more advanced number theory. In particular, by constructing integer relations by factoring elements in a number field, this reduces the size of the integer relations to be tested for smoothness, thus increasing the probability that a candidate relation is smooth, and resulting in less work to find the desired number of relations. Decreasing the size of the integers to be tested makes a dramatic difference overall: as a concrete example, 0.5% of 32-bit integers are 8-bit smooth, but only 0.0025% of 48-bit integers are.

This results in an algorithm divided into five stages: *polynomial selection*, in which one chooses a number field to work in, followed by *relation collection*, a *filtering* stage in which one reduces the size of the next phase, *linear algebra*, and a final phase called the *square root* phase in the factoring context (see Figure 1). The relation collection and linear algebra phases, which are described in general terms above, occupy the bulk of the running time.

1.2 Asymptotic running time

The running time of GNFS is not easy to express. Heuristically, it is possible to derive an asymptotic formula from coarse asymptotic estimates of the number of smooth integers. The cost of factoring N with GNFS, as N grows, is

$$\exp\left((64/9)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}(1 + o(1))\right).$$

This running time is *subexponential*, meaning that it is faster than a purely exponential-time algorithm in $\log N$ would be, but slower than a polynomial-time algorithm.

However, the $1 + o(1)$ factor hides a significant amount of performance for concrete parameters. An algorithmic improvement can bring a polynomial speedup to the algorithm (that is, a polynomial in $\log N$), without leaving this broad complexity class. While this general asymptotic running time for the number field sieve has not been improved since the algorithm was developed in the early 1990s, there has been algorithmic progress that has had a dramatic impact on the actual running time of the algorithm, all of which is hidden by the "$o(1)$" term.

The recommended key sizes for RSA in practice are selected based on this asymptotic estimate of running time. This subexponential running time of the NFS algorithm is why RSA key sizes are so much larger than symmetric key sizes at the same security level. The best attacks against the AES cipher take only slightly less than 2^{128} time for a 128-bit key, so 128-bit AES is expected to provide nearly 128 bits of security. However, for RSA, the usual estimate is that in order to have 128 bits of security, one should use at least a 3072-bit modulus.

1.3 Practical progress

Although the general structure of the number field sieve algorithm has remained roughly the same for 30 years, there have been many refinements during that time that have improved performance by significant constant factors. At a high level, the number field sieve is a complex algorithm with a large number of adjustable parameters that affect algorithm stages in different and not always well understood ways.

Below, we touch on a handful of recent refinements to the most time-consuming steps of NFS: polynomial selection, relation collection, and linear algebra.

In the polynomial selection step, an implementation uses analytic estimates and experimental sampling to find a good choice of number field. The properties of the chosen polynomial have a significant effect on the overall running time of the algorithm, so it can be worthwhile to expend effort on this step. Our recent 795-bit and 829-bit factoring records took advantage of improvements to polynomial selection to achieve relatively faster running times than extrapolations from previous records would suggest [3]. Yet there is reason to expect that further progress is possible: if "ideal" polynomials in the sense of [5] could be found, relation collection could be made almost twice as fast. An efficient method of finding such "ideal" polynomials remains an open problem.

The relation collection step is generally the computationally most expensive part of the NFS algorithm. Fortunately, it is embarrassingly parallel, and recent computations have been able to make extensive use of idle time on large clusters. There are a variety of algorithms that can be applied to this problem, which can vary in speed and memory requirements. The recent factoring records spent a good deal of effort in exploring the relative benefits of many of these algorithms, some of which had never been used before. The choice and parameterization of algorithms in relation collection also interacts with the NFS algorithm in general (with linear algebra, for example), which necessitates some fine-tuning. In our latest record computations, this fine-tuning work was supported by extensive use of simulations to estimate the running time of each stage of NFS and optimize the complex interactions between parameters, which proved tremendously useful.

Historically, the linear algebra phase was believed to be a major barrier to scaling factoring to cryptographically relevant numbers because of prohibitively large memory requirements. It was also believed that this phase could not effectively be parallelized, and thus it would be a bottleneck. Research progress has shown that both of these statements are false. The block Wiedemann algorithm, which is used for NFS sparse linear algebra, has evolved both algorithmically and in practice into a very versatile tool. It can effectively be parallelized, and recent record computations parallelized this phase across multiple remote computational resources. And on modern computing systems, the memory requirements do not seem to be limiting the scale of current computations at all.

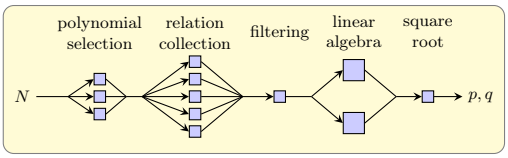
All of these refinements are specific to the number field sieve algorithm. A brand new algorithm might yet be discovered in the future, which could completely change the integer factoring landscape.

1.4 The state of the art

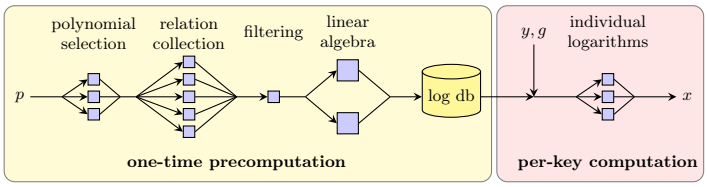
By convention, factoring records are often measured using the RSA challenges published by RSA Labs in 1991, although the cash prizes expired years ago.

The latest factoring (and discrete logarithm) records are depicted in Figure 2, culminating with the factorization of RSA-250 (a 829-bit composite number) in February 2020, for a computation cost of about 2,700 core-years.

These academic results are far from the limit of feasibility, especially for a state-level attacker. It is obvious that larger computations are possible, and it is widely accepted that 1024-bit factoring and discrete logarithm



The Number Field Sieve algorithm for factoring is a complex multi-stage algorithm in which each stage has different computing requirements and potential for parallelization, as well as numerous tuneable parameters that enable different tradeoffs.



The number field sieve algorithm for discrete logarithm is closely related to the NFS algorithm for factoring.

Figure 1: Main steps of NFS and NFS-DL.

are feasible *now*, with hardware and software that currently exist. A ballpark estimate is that factoring a 1024-bit key would be about 200 times harder than RSA-250, or around 500,000 core-years of computation. The associated sparse matrix is expected to have about 4 billion rows and columns. Dealing with such a matrix would mobilize tens of thousands of CPU cores over several months, which is certainly feasible. The challenge of computing discrete logarithms for this size is a step further in terms of difficulty and resource requirements, but would still be feasible. Nobody has done such a computation yet in public, since academic research teams don't have access to that much computing power. From an academic standpoint, carrying out such a computation would not surprise the research community. However, carrying out these types of concrete cryptanalytic computations appears to be necessary to encourage users to move to stronger key sizes and algorithms in the real world. Recent history has shown that in the context of hash functions, work on rogue MD5 certificates [11] or the computation of actual collisions on SHA-1 [10] were both instrumental in the deprecation of these insecure hash functions.

2 Finite field discrete logarithms and Diffie-Hellman

In the discrete logarithm problem over prime fields, g and p are public integers, and the problem is, given some input $y = g^x \bmod p$, to find the discrete logarithm x.

More generally, we can consider the discrete logarithm problem over finite fields $\mathrm{GF}(p^n)$ of p^n elements, where p is a prime (the *characteristic*) and n is an integer (the *extension degree*). Mathematically, there are three general algorithmic classes: the prime field case $\mathrm{GF}(p)$ where $n = 1$, the small-characteristic case where p is very small (typically $p = 2$ or $p = 3$), and the medium characteristic, which lies between these two extremes.

The security of Diffie-Hellman key exchange is "based on" the discrete logarithm problem in the sense that if an attacker can compute the discrete logarithm of one of the publicly transmitted key exchange values, they can use this value to easily compute the shared secret. However, like the relationship between factoring and RSA, it is not known whether the two problems are equivalent: although discrete logarithm is the best attack known to solve the Diffie-Hellman problem, there may be some yet-undiscovered algorithm to compute the shared secret that does not reveal the discrete logarithm of the public values.

The discrete logarithm problem seems to be closely related to the problem of integer factoring, and a number of algorithms for solving integer factorization have analogues to solve the discrete logarithm problem.

Strangely, this eerie connection between the two problems does not seem to have been formally understood.

2.1 Discrete logarithms in prime fields

The most common case of discrete logarithm-based cryptography takes place when the public modulus p is a prime, and g is a generator of a prime-order multiplicative subgroup modulo p. As in the case of RSA, the prime structure and size of these parameters are chosen to avoid a number of elementary discrete logarithm algorithms that work for small, composite-order, or otherwise structured groups like the Pollard rho algorithm, parallel collision search, and Pohlig-Hellman algorithm.

Once these elementary algorithmic attacks have been thwarted by proper parameter choices, the best algorithm in general to solve the discrete logarithm problem is the number field sieve algorithm for discrete logarithms, or NFS-DL for short. Its complexity depends on the modulus p. The asymptotic running time for NFS to factor an integer N is the same as the asymptotic running time for NFS-DL for a modulus p of the same size as N.

The NFS-DL algorithm, like the NFS algorithm for factoring, has five main stages: polynomial selection, sieving, filtering, linear algebra, and finally the computation of individual logarithms (see Figure 1).

The first three stages have a similar structure to the analogous algorithm for factoring, but there are some important differences. In the polynomial selection phase, the fact that the general setup of NFS-DL targets a field instead of a ring permits an improved strategy that yields much better results for current record computational sizes: in our most recent record, this allowed a 5-fold speedup [3].

Second, the linear system to be solved is no longer modulo 2, but instead modulo a larger prime, in particular the order of the subgroup generated by g. This makes linear algebra more expensive than in the factoring case. The final difference is that polynomial selection, relation collection, and linear algebra depend only on the prime modulus p, and not on the target value to compute the discrete logarithm of. This is different from the case of factoring, where all stages of the NFS algorithm depend on the integer N to be factored, and means that an attacker could carry out the most expensive parts of a discrete logarithm computation offline, and reuse them for many different key exchanges using the same parameters [1].

2.2 Current records and comparison with factoring

The current record for largest discrete logarithm computation in a prime field was done modulo a 795-bit prime [3]. This computation took 3,100 core-years and little over 1 calendar year of computational time.

While the asymptotic running time of the NFS algorithm is the same for discrete logarithm and factoring, discrete logarithm has generally been perceived as somewhat more difficult, mainly because of the increased cost for the linear algebra phase. This can be seen empirically in Figure 2, as discrete logarithm records have historically lagged several years behind the size of record factorizations. However, recent computations have shown that the improved flexibility in polynomial selection can result in much closer practical running times: for 795-bit factoring and discrete logarithms, [3] shows that computing discrete logarithms is about three times harder than factoring.

2.3 The small characteristic case

One can also consider the problem of computing discrete logarithms in finite fields of order 2^n and 3^n, or more generally p^n for small p. It has a close relationship to the discrete logarithm problem in "large-characteristic" fields of order p described above. For more than two decades, the best algorithm to solve discrete logarithm in the small-characteristic case was the function field sieve, an algorithm closely related to the number field sieve, and the best estimates of its asymptotic running time were similar to those for large-characteristic discrete logarithms using the number field sieve.

However, starting in 2013, a series of algorithmic advances led to dramatic improvements in running time for the small characteristic case, culminating in new algorithms of *quasi-polynomial* ($n^{O(\log n)}$) running-time [2]. (Remarkably, this line of research has recently led to a *proven* algorithm [6] achieving quasi-polynomial complexity, a unique feat among algorithms for factoring and discrete logarithms in finite fields.) The

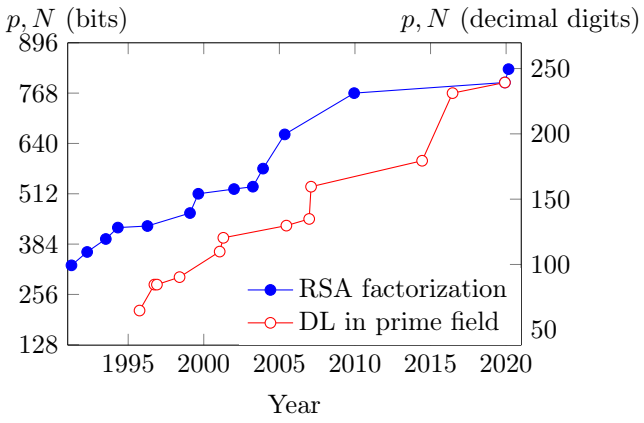


Figure 2: RSA factorization and DL computation records.

immediate consequence of this breakthrough was that the discrete logarithm problem in fields of the form $\mathrm{GF}(2^n)$ was no longer hard enough for cryptography. While these fields had rarely been considered for cryptographic use, their demise also discarded other cryptographic constructions that implicitly relied on their hardness, such as supersingular elliptic curves over binary fields, which used to be part of the zoo of curves potentially useful in pairing-based cryptography.

The specific advances used to achieve these results do not seem to apply to the large-characteristic case. At the same time, as in the case of RSA, parameter sizes for discrete logarithm-based cryptography including Diffie-Hellman key exchange, DSA digital signatures, and ElGamal encryption are set based on extrapolations of the running time of NFS-DL for large-characteristic fields. The main evidence for the security of these parameter choices is the lack of asymptotic improvements in the large characteristic case during the last three decades. However, as the significant improvement in the small-characteristic case shows, previously overlooked improvements can be found after decades of stasis.

2.4 Discrete logarithms in small to moderate degree extension fields

Between the "easy case" of fields of extremely small characteristic and the more challenging case of prime fields detailed above, the discrete logarithm problem in extension fields is also an interesting research area. This topic is of particular interest to pairing-based cryptography, which implicitly relies on the hardness of the DLP in moderate-degree extensions of prime fields. In turn, this problem also matters for some cryptocurrency applications that use succinct and efficient zero-knowledge proofs built from elliptic curve pairings.

Special variants of NFS are well adapted in this case, notably the "Tower" Number Field Sieve (TNFS). Several new algorithms have emerged in recent years, and practical progress has been steady. The current record is a 521-bit computation in a degree-6 field [4]. As a result of this ongoing progress, accurate assessments of feasibility limits for the DLP in such fields are much harder to obtain, as they rely on fragile and fast-changing empirical evidence.

3 Elliptic curve discrete logarithms

The discrete logarithm problem on elliptic curve (ECDLP) has seen considerably less cryptanalytic progress than factoring or finite-field discrete logarithms. The case of highest cryptographic interest (for elliptic curve Diffie-Hellman or elliptic curve digital signature algorithms, for example) is that of ordinary elliptic curves of almost-prime group order over a prime field.

Elliptic curve parameters for cryptography are generally chosen to avoid attacks via particular structures in elliptic curves, or generic group attacks like the Pohlig-Hellman algorithm that exploit composite group

orders. Thus for cryptographically relevant curves, the best cryptanalytic algorithms are the Pollard Rho or parallel collision search algorithms. The expected runtime of these algorithms is well known, and equal to $O(\sqrt{n})$ elliptic curve operations, where n is the order of the group, and the $O()$ constant can be made explicit. This runtime is exponential in the size of the public key. Since elliptic curve parameter sizes for cryptography are chosen based on the running time of the best attacks, this is why cryptography using a 256-bit elliptic curve is considered to offer 128 bits of security. The lack of faster attacks for well-constructed curves is why elliptic curve cryptography offers much shorter keys, and thus improved performance, over RSA or finite-field discrete-logarithm based cryptography.

The progress of records for ECDLP cryptanalysis has remained fairly predictable over the past several decades. The current computational record is 114 bits, with progress attributed largely to Moore's law, improvements in hardware, and fast implementations.

Since the current best discrete logarithm algorithms for ECDLP over cryptographically relevant curves are algorithms that apply to any generic cyclic group, it remains an open question whether there is some algorithmic approach that can somehow take advantage of algebraic structure in order to achieve improved cryptanalytic performance. However attempts to mount "index-calculus"-like algorithms (as in NFS) for ECDLP have led to mixed results. No overarching structure like the number field structure in NFS has yet been able to be exploited in this case. Research has proven that ECDLP over composite degree extension fields must be avoided. Beyond this case, some *ad hoc* algorithms have been proposed. However, bar some claims relying on unconvincing heuristics, they have not been proven to have greater cryptanalytic potential than the generic algorithms.

Over binary fields of prime extension degree, no dramatic progress akin to the quasi-polynomial time algorithm for DLP in $\mathrm{GF}(2^n)$ has appeared. As such, current ECDLP records in this case are also in the same size range as for curves over a prime field.

Pairing-friendly curves are also an interesting special case. By construction, these curves allow one to transfer the elliptic curve DLP to a finite field, whose size is dependent on the curve being chosen. This gives two distinct avenues for cryptanalysis. On the elliptic curve side, the problem is mostly identical to the case of general curves (with mostly identical practical scaling as well), with the exception that automorphisms might reduce the security level by one bit. On the finite field size, recent developments show that some pairing-friendly curve choices might provide a security level that is several bits short of the claimed security level. This area is currently evolving with the practical developments of NFS-like algorithms for moderate-size extension fields (see §2.4).

4 Prospects for the future

We are at the cusp of a great transition in public-key cryptographic algorithms. All of the computational number theory problems described above that appear to be challenging to solve with a classical computer can, in theory, be broken in polynomial time on a quantum computer. However, it remains an open question whether scaling physical quantum computers to the sizes and error tolerances necessary to run Shor's algorithm can ever be achieved [9].

Efforts to standardize new "post-quantum" public-key cryptographic algorithms are underway, and will result in algorithm recommendations within the next few years. Future adoption of these algorithms is likely inevitable even if a quantum computer capable of Shor's algorithm is never built. These new algorithms will have a very different mathematical structure from the existing common public-key cryptographic algorithms, opening the door to decades of new research in cryptanalysis.

In the meantime, our existing public-key algorithms will continue to be used, and may have a long half-life when used in conjunction with these newer post-quantum algorithms. Research on classical cryptanalysis will remain relevant because of the long lifetimes of cryptographic algorithms, and because mathematical breakthroughs can result in fast progress. While there is currently no consensus among experts on a timeline for building a quantum computer capable of implementing Shor's algorithm, or whether such a quantum computer will *ever* be feasible to build, experts do agree that the sheer scale of the engineering effort required is large enough that it would not likely be possible for a single government, say, working alone, to achieve

such a feat entirely in secret in a short timeline and take the world by surprise. [9] In contrast, progress in classical cryptanalysis sometimes requires as little as a single mathematical insight and some software development. We have no evidence that the current running time of the number field sieve is optimal for factoring and discrete logarithm, and as we have seen in the small-characteristic discrete logarithm case, practical mathematical progress can happen rapidly after new insights even after decades of stability, taking even experts by surprise.

References

[1] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 5–17. ACM Press (Oct 2015). doi:10.1145/2810103.2813707

[2] Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer (May 2014). doi:10.1007/978-3-642-55220-5_1

[3] Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 62–91. Springer (Aug 2020). doi:10.1007/978-3-030-56880-1_3

[4] De Micheli, G., Gaudry, P., Pierrot, C.: Lattice enumeration for Tower NFS: a 521-bit discrete logarithm computation. Cryptology ePrint Archive, Report 2021/707 (2021), https://ia.cr/2021/707

[5] Kleinjung, T.: Polynomial Selection for the Number Field Sieve. In: Bos, J.W., Lenstra, A.K. (eds.) Topics in Computational Number Theory Inspired by Peter L. Montgomery, pp. 161–174. Cambridge University Press (2017). doi:10.1017/9781316271575.007

[6] Kleinjung, T., Wesolowski, B.: Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic. J. Amer. Math. Soc. (Sep 2021). doi:10.1090/jams/985

[7] Lenstra, A.K., Lenstra, Jr., H.W., Manasse, M.S., Pollard, J.M.: The number field sieve (extended abstract). In: Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC). p. 564–572. ACM Press, Baltimore, MD (1990). doi:10.1145/100216.100295

[8] Lenstra, A.K., Lenstra, Jr., H.W. (eds.): The development of the number field sieve, Lecture Notes in Math., vol. 1554. Springer (1993). doi:10.1007/BFb0091534

[9] National Academies of Sciences, Engineering, and Medicine: Quantum Computing: Progress and Prospects. The National Academies Press, Washington, DC (2019). doi:10.17226/25196

[10] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 570–596. Springer (Aug 2017). doi:10.1007/978-3-319-63688-7_19

[11] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer (Aug 2009). doi:10.1007/978-3-642-03356-8_4