11.6 Discrete logarithms over finite fields

Andrew Odlyzko, University of Minnesota

Surveys and detailed expositions with proofs can be found in [7, 25, 26, 28, 33, 34, 47].

11.6.1 Basic definitions

11.6.1 Remark Discrete exponentiation in a finite field is a direct analog of ordinary exponentiation. The exponent can only be an integer, say n, but for w in a field F, w^n is defined except when $w = 0$ and $n \leq 0$, and satisfies the usual properties, in particular $w^{m+n} = w^m w^n$ and (for u and v in F) $(uv)^m = u^m v^m$. The discrete logarithm is the inverse function, in analogy with the ordinary logarithm for real numbers. If F is a finite field, then it has at least one primitive element g; i.e., all nonzero elements of F are expressible as powers of g, see Chapter **??**.

11.6.2 Definition Given a finite field F, a primitive element g of F, and a nonzero element w of F, the *discrete logarithm* of w to base g, written as $\log_g(w)$, is the least non-negative integer n such that $w = g^n$.

11.6.3 Remark The value $\log_g(w)$ is unique modulo $q - 1$, and $0 \leq \log_g(w) \leq q - 2$. It is often convenient to allow it to be represented by any integer n such that $w = g^n$.

11.6.4 Remark The discrete logarithm of w to base g is often called the *index* of w with respect to the base g. More generally, we can define discrete logarithms in groups. They are commonly called *generic discrete logs*.

11.6.5 Definition If G is a group (with multiplication as group operation), and g is an element of G of finite order m, then for any element h of $\langle g \rangle$, the cyclic subgroup of G generated by g, the *discrete logarithm* of h to base g, written as $\log_g(h)$, is the least non-negative integer n such that $h = g^n$ (and therefore $0 \leq \log_g(h) \leq m - 1$).

11.6.6 Remark The definition of a group discrete log allows for consideration of discrete logs in finite fields when the base g is not primitive, provided the argument is in the group $\langle g \rangle$. This situation arises in some important applications, in particular in the U.S. government standard for the Digital Signature Algorithm (DSA). DSA operations are performed in a field F_p with p a prime (nowadays recommended to be at least 2048 bits). This prime p is selected so that $p - 1$ is divisible by a much smaller prime r (specified in the standard to be of 160, 224, or 256 bits), and an element h of F_p is chosen to have multiplicative order r (say by finding a primitive element g of F_p and setting $h = g^{(p-1)/r}$). The main element of the signature is of the form h^s for an integer s, and ability to compute s would break DSA. DSA can be attacked either by using generic finite group discrete log algorithms in the group $\langle h \rangle$ or finite field algorithms in the field F_p (which can then easily yield a solution in $\langle h \rangle$).

11.6.7 Remark The basic properties of discrete logs given below, such as the change of base formula, apply universally. On the other hand, many of the discrete log algorithms described later are valid only in finite fields. Generally speaking, discrete logs are easiest to compute in finite fields, since they have a rich algebraic structure that can be exploited for cryptanalytic purposes. Much of the research on discrete logs in

other settings has been devoted to embedding the relevant groups inside finite fields in order to apply finite field discrete log algorithms.

11.6.8 Remark This section is devoted to finite field discrete logs, and only gives a few references to other ones. For elliptic curve discrete logs, the most prominent collection, see Section **??**. However, other groups have also been used, for example class groups of number fields [27].

11.6.2 Modern computer implementations

11.6.9 Remark Most popular symbolic algebra systems contain some implementations of discrete log algorithms. For example, Maple has the *mlog* function, while Mathematica has *FieldInd*. More specialized systems for number theoretic and algebraic computations, such as Magma, PARI, and Sage, also have implementations, and typically can handle larger problems. Thus for all but the largest problems that are at the edge of computability with modern methods, widely available and easy to use programs are sufficient. Tables of finite fields, such as those in [25], are now seldom printed in books.

11.6.3 Historical remarks

11.6.10 Remark Until the mid-1970s, the main applications for discrete logs were similar to those of ordinary logs, namely in routine computations, but this time in finite fields. They allowed replacement of relatively hard multiplications by easier additions. What was frequently used was *Zech's logarithm* (also called *Jacobi's logarithm*, cf. [25]), which is a modification of the ordinary discrete log. In a finite field F with primitive element g, Zech's log of an integer n is defined as the integer $Z(n)$ mod $(q-1)$ which satisfies $g^{Z(n)} = 1 + g^n$. This provides a quick way to add elements given in terms of their discrete logs: aside from boundary cases, $g^m + g^n = g^m(1 + g^{n-m}) = g^{m+Z(n-m)}$.

11.6.11 Remark As with ordinary logarithms, where slide rules and log tables have been replaced by calculators, such routine applications of discrete logs in small or moderately large fields now rely on computer algebra systems.

11.6.12 Remark Interest in discrete logs jumped dramatically in the mid-1970s with the invention of public key cryptography, see Chapter **??**. While discrete exponentiation is easy, the discrete logarithm, its inverse, appeared hard, and this motivated the invention of the Diffie–Hellman key exchange protocol, the first practical public key cryptosystem. Efficient algorithms for discrete logs in the field over which this protocol is implemented would make it insecure.

11.6.13 Remark The Diffie–Hellman problem is to compute g^{xy}, the key that the two parties to the Diffie–Hellman protocol obtain, from the g^x and g^y that are visible to the eavesdropper. Although this problem has attracted extensive attention, it has not been solved, and for the most important cases of finite field and elliptic curve discrete logs, it is still unknown whether the Diffie–Hellman problem is as hard as the discrete log one. See [4] for recent results and references.

11.6.14 Remark It is known that single bits of discrete logs are about as hard to compute as the entire discrete logs [14].

11.6.15 Remark There are some rigorous lower bounds on discrete log problems, but only

for groups given in ways that restrict what can be done in them [31, 45].

11.6.16 Remark Various cryptosystems other than the Diffie-Hellman one have been proposed whose security similarly depends on the intractability of the discrete log problem, see Section **??**. Many of them can be used in settings other than finite fields.

11.6.17 Remark There are close analogies between integer factorization and discrete logs in finite fields, and most (but not all) of the algorithms in one area have similar ones in the other. This will be seen from some of the references later. In general, considerably less attention has been devoted to discrete logs than to integer factorization. Hence the smaller sizes of discrete log problems that have been solved result both from the greater technical difficulty of this problem as compared to integer factorization and from less effort being devoted to it.

11.6.18 Remark Peter Shor's 1994 result [44] shows that if quantum computers become practical, discrete logs will become easy to compute. Therefore cryptosystems based on discrete logs may all become suddenly insecure.

11.6.4 Basic properties of discrete logarithms

11.6.19 Remark Suppose that G is a group, and g an element of finite order m in G. If u and v are two elements of $\langle g \rangle$, then

$$\log_g(uv) \equiv \log_g(u) + \log_g(v) \qquad (\text{mod } m),$$

$$\log_g(u^{-1}) \equiv -\log_g(u) \qquad (\text{mod } m).$$

11.6.20 Remark (**Change of base formula**): Suppose that G is a group, and that g and h are two elements of G that generate the same cyclic subgroup $\langle g \rangle = \langle h \rangle$ of order m. If u is an element of $\langle g \rangle$, then

$$\log_g(u) \equiv \log_h(u) * \log_g(h) \qquad (\text{mod } m),$$

and therefore

$$\log_g(h) \equiv 1/\log_h(g) \qquad (\text{mod } m).$$

These formulas mean that one can choose the most convenient primitive element to work with in many applications. For example, in finite fields F_{2^k}, elements are usually represented as polynomials with binary coefficients, and one can find (as verified by experiment and inspired by heuristics, but not proved rigorously) primitive elements that are represented as polynomials of very low degree. This can offer substantial efficiencies in implementations. However, it does not affect the security of the system. If discrete logs are easy to compute in one base, they are easy to compute in other bases. Similarly, the change of the irreducible polynomial that defines the field has little effect on difficulty of the discrete log problem.

11.6.5 Chinese Remainder Theorem reduction: The Silver–Pohlig–Hellman algorithm

11.6.21 Remark If the order of the element g can be factored even partially, the discrete log problem reduces to easier ones. This is the Silver–Pohlig–Hellman technique [36]. Suppose that g is an element of finite order m in a group G, and m factors

as $m = m_1 m_2$ with $\gcd(m_1, m_2) = 1$. Then the cyclic group $\langle g \rangle$ is the direct product of the cyclic groups $\langle g^{m_2} \rangle$ and $\langle g^{m_1} \rangle$ of orders m_1 and m_2, respectively. If we determine $a = \log_{g^{m_2}}(w^{m_2})$ and $b = \log_{g^{m_1}}(w^{m_1})$, the Chinese Remainder Theorem tells us that $\log_g(w)$ is determined completely, and in fact we obtain

$$\log_g(w) \equiv b * x * m_1 + a * y * m_2 \qquad (\text{mod } m),$$

where x and y come from the Euclidean algorithm computation of $\gcd(m_1, m_2)$, namely $1 = xm_1 + ym_2$. This procedure extends easily to more than two relatively prime factors.

11.6.22 Remark When m, the order of g, is a prime power, say $m = p^k$, the computation of $\log_g(w)$ reduces to k discrete log computations in a cyclic group of p elements. For example, if $r = p^{k-1}$ and $h = g^r$, $u = w^r$, then h has order p, and computing $\log_h(u)$ yields the reduction of $\log_g(w) \pmod p$. This process can then be iterated to obtain reduction modulo p^2, and so on.

11.6.23 Remark The above remarks, combined with results of the next section, show that when the complete factorization of the order of g can be obtained, discrete logs can be computed in not much more than $r^{1/2}$ operations in the group, where r is the largest prime in the factorization.

11.6.24 Remark In a finite field, any function can be represented by a polynomial. For the discrete log, such polynomials do turn out to have some esthetically pleasing properties, see [30, 32, 50, 51]. However, so far they have turned out to be of no practical use whatever.

11.6.6 Baby steps–giant steps algorithm

11.6.25 Remark We next consider some algorithms for discrete logs that work in very general groups. The basic one is the baby steps–giant steps method that combines time and space, due to D. Shanks [43].

11.6.26 Algorithm Baby steps–giant steps algorithm: Suppose that G is a group and g is an element of G of finite order m. If $h \in \langle g \rangle$, $h = g^k$, and $w = \lceil m^{1/2} \rceil$, then k can be written as $k = aw + b$ for some (often non-unique) $a,\ b$ with $0 \le a,\ b < w$. To find such a representation, compute the set $A = \{g^{jw} : 0 \le j < w\}$ and sort it. This takes $m^{1/2} + O(\log(m))$ group operations and $O(m^{1/2} \log(m))$ sorting steps, which are usually very easy, since they can be performed on bit strings, or even initial segments of bit strings. Next, for $0 \le i < w$, compute hg^{-i} and check whether it is present in A. When it is, we obtain the desired representation $k = jw + i$.

11.6.27 Remark The baby steps–giant steps technique has the advantage of being fully deterministic. Its principal disadvantage is that it requires storage of approximately $m^{1/2}$ group elements. A space-time tradeoff is available, in that one can store a smaller list (the set A in the notation above, with fewer but larger "giant steps") but then have to do more computing (more "baby steps").

11.6.28 Remark The baby steps–giant steps algorithm extends easily to many cases where the discrete log is restricted in some way. For example, if it is known that $\log_g(w)$ lies in an interval of length n, the basic approach sketched above can be modified to find it in $O(n^{1/2})$ group operations (plus the usual sorting steps). Similarly, if the discrete log k is allowed to have only small digits when represented in some

base (say binary digits in base 10), then the running time will be about the square root of the number of possibilities for k. For some other recent results, see [46].

11.6.7 Pollard rho and kangaroo methods for discrete logs

11.6.29 Remark In 1978, J. Pollard invented two randomized methods for computing discrete logs in any group, the rho method, and the kangaroo (or lambda) technique [37]. Just like Pollard's earlier rho method for integer factorization, they depend on the birthday paradox, which says that if one takes a random walk on a completely connected graph of n vertices, one is very likely to revisit the same vertex in about $n^{1/2}$ steps. These discrete log algorithms also depend, just as the original rho method does, on the Floyd algorithm (Section 3.1 of [23]) for detecting cycles with little memory at some cost in running time, in that they compare x_{2i} to x_i, where x_i is the position of the random walk at time i.

11.6.30 Remark Since the rho and kangaroo methods for discrete logs are probabilistic, they cannot guarantee a solution, but heuristics suggest, and experiments confirm, that both run in expected time $O(m^{1/2})$, where m is the order of the group. This is the same computational effort as for the baby steps–giant steps algorithm. However, the rho and kangaroo methods have two advantages. One is that they use very little memory. Another one is that, as was first shown by P. van Oorschot and M. Wiener [49], they can be parallelized, with essentially linear speedup, so that k processors find a solution about k times faster than a single one. We sketch just the standard version of the rho method, and only briefly.

11.6.31 Algorithm Rho algorithm for discrete logs: Partition the group $\langle g \rangle$ of order m into three roughly equal sets S_1, S_2, and S_3, using some property that is easy to test, such as the first few bits of a canonical representation of the elements of G. To compute $\log_g(h)$, define a sequence $w_0, w_1, \ldots$ by $w_0 = g$ and for $i > 0$, $w_{i+1} = {w_i}^2, w_i g$, or $w_i h$, depending on whether $w_i \in S_1, S_2$, or S_3. Then each w_i is of the form

$$w_i = g^{a_i} h^{b_i}$$

for some integers a_i, b_i. If the procedure of moving from w_i to w_{i+1} behaves like a random walk (as is expected), then in $O(m^{1/2})$ steps we will find i such that $w_i = w_{2i}$, and this will give a congruence

$$a_i + b_i \log_g(h) \equiv a_{2i} + b_{2i} \log_g(h) \pmod{m}.$$

Depending on the greatest common divisor of m and $b_i - b_{2i}$ this congruence will typically either yield $\log_g(h)$ completely, or give some stringent congruence conditions, which with the help of additional runs of the algorithm will provide a complete solution.

11.6.32 Remark The low memory requirements and parallelizability of the rho and kangaroo algorithms have made them the methods of choice for solving general discrete log problems. There is a substantial literature on various modifications, although they do not improve too much on the original parallelization observations of [49]. Some references are [6, 20, 38, 48].

11.6.33 Remark The rho method, as outlined above, requires knowledge of the exact order m of the group. The kangaroo method only requires an approximation to m. The kangaroo algorithm can also be applied effectively when the discrete log is known to lie in a restricted range.

11.6.8 Index calculus algorithms for discrete logs in finite fields

11.6.34 Remark The rest of this section is devoted to a brief overview of index calculus algorithms for discrete logs. Unlike the Shanks and Pollard methods of the previous two subsections, which take exponential time, about $m^{1/2}$ for a group of order m, the index calculus techniques are subexponential, with running times closer to $\exp((\log(m))^{1/2})$ and even $\exp((\log(m))^{1/3})$. However, they apply directly only to finite fields. That is why much of the research on discrete logs in other groups of cryptographic interest, such as on elliptic curves, is devoted to finding ways to reduce those problems to ones in finite fields.

11.6.35 Remark In the case of DSA mentioned at the beginning of this section, the recommended size of the modulus p has increased very substantially, from 512 to 1024 bits when DSA was first adopted, to the range of 2024 to 3036 bits more recently. The FIPS 186-3 standard specifies bit lengths for the two primes p and r of (1024, 160), (2048, 224), (2048, 256), and (3072, 256). The relative sizes of p and r were selected to offer approximately equal levels of security against index calculus algorithms (p) and generic discrete log attacks (r). The reason for the much faster growth in the size of p is that with the subexponential running time estimates, the effect of growing computing power is far more pronounced on the p side than on the r side. In addition, while there has been no substantial theoretical advance in index calculus algorithms in the last two decades, there have been numerous small incremental improvements, several cited later in more detailed discussions. On the other hand, there has been practically no progress in generic discrete log algorithms, except for parallelization.

11.6.36 Remark The basic idea of index calculus algorithms dates back to Kraitchik, and is also key to all fast integer factorization algorithms. In a finite field F with $|F| = q$, and with primitive element g, if we find some elements $x_i, y_j \in F$ such that

$$\prod_{i=1}^{r} x_i = \prod_{j=1}^{s} y_j,$$

then

$$\sum_{i=1}^{r} \log_g x_i \equiv \sum_{j=1}^{s} \log_g y_j \qquad (\text{mod } q-1).$$

If enough equations are collected, this linear system can be solved for the $\log_g x_i$ and $\log_g y_j$. Singular systems are not a problem in practice, since typically computations generate considerably more equations than unknowns, and one can arrange for g itself to appear in the multiplicative relations.

11.6.37 Remark To compute $\log_g w$ for some particular $w \in F$ with index calculus algorithms, it is often necessary to run a second stage that produces a relation involving w and the previously computed discrete logs. In some algorithms the second stage is far easier than the initial computation, in others it is of comparable difficulty.

11.6.38 Remark For a long time (see [33] for references), the best index calculus algorithms for both integer factorization and discrete logs had running times of the form

$$\exp((c+o(1))(\log q)^{1/2}(\log\log q)^{1/2}) \quad \text{as} \quad p \to \infty$$

for various constants $c > 0$, where q denotes the integer being factored or the size of the finite field. The first practical method that broke through this running time barrier was Coppersmith's algorithm [9] for discrete logs in fields of size $q = 2^k$ (and more generally, of size $q = p^k$ where p is a small prime and k is large). It had running time of approximately

$$\exp(C(\log q)^{1/3}(\log\log q)^{2/3}),$$

where the C varied slightly, depending on the distance from k to the nearest power of p, and in the limit as $k \to \infty$ it oscillated between two bounds [33]. The function field sieve of Adleman [1], which also applies to fields with $q = p^k$ where p is relatively small, improves on the Coppersmith method, but has similar asymptotic running time estimate. For the latest results on its developments, see [17, 19, 39].

11.6.39 Remark The running time of Coppersmith's algorithm turned out to also apply to the number field sieve. This method, which uses algebraic integers, was developed for integer factorization by Pollard and Hendrik Lenstra, with subsequent contributions by many others. It was adopted for discrete log computations in prime fields by Gordon [13], with substantial improvements by other reseachers. For the latest estimates and references, see [8, 18, 40, 41].

11.6.9 Smooth integers and smooth polynomials

11.6.40 Remark The index calculus algorithms depend on a multiplicative splitting of some elements, such as integers or polynomials, into such elements drawn from a smaller collection. This smaller collection usually is made up of elements that by some measure (norm) are small. The essence of index calculus algorithms is to select general elements from the large set at random, but as intelligently as possible in order to maximize the chances they will have the desired type of splitting. Usually elements that do have such splittings are called "smooth."

11.6.41 Remark There are rigorous analyses that provide estimates of how often elements in various domains are "smooth." For ordinary integers, there are the estimates of [15]. For algebraic integers, we can use [5, 42]. For polynomials over finite fields, recent results are [35].

11.6.10 Sparse linear systems of equations

11.6.42 Remark Index calculus algorithms for discrete logs require the solution of linear equations modulo $q - 1$, where q is the size of the field. As in the Silver–Pohlig–Hellman method, the Chinese Remainder Theorem (and an easy reduction of the case of a power of a prime to that of the prime itself) reduces the problem to that of solving the system modulo primes r that divide $q - 1$. (For more extensive discussion of linear algebra over finite fields, see Section **??**.)

11.6.43 Remark The linear algebra problems that arise in index calculus algorithms for integer factorization are very similar, but simpler, in that they are all just mod 2. For discrete log problems to be hard, they have to be resistant to the Silver–Pohlig–Hellman attack. Hence $q-1$ has to have at least one large prime factor r, and so the linear system has to be solved modulo a large prime. That increases the complexity of the linear solution computation, and thus provides slightly higher security for discrete log cryptosystems.

11.6.44 Remark A key factor that enables the solution of the very large linear systems that arise in index calculus algorithms is that these systems are very sparse. (Those "smooth" elements do not involve too many of the "small" elements in the multiplicative relations.) Usually the structured gaussian elimination method (proposed in [33] and called there intelligent gaussian elimination, afterwards renamed in the first practical demonstration of it [24], now sometimes called filtering) is applied first. It combines the relations in ways that reduce the system to be solved and do not destroy the sparsity too far. Then the conjugate gradient, the Lanczos, or the Wiedemann methods (developed in [12, 52], the first two demonstrated in practice in [24]) that exploit sparsity are used to obtain the final solution.

11.6.45 Remark For the extremely very large linear systems that are involved in record-setting computations, distributed computation is required. The methods of choice, once structured gaussian elimination is applied, are the block Lanczos and block Wiedemann methods [10, 11, 29].

11.6.46 Remark Some symbolic algebra systems incorporate implementations of the sparse linear system solvers mentioned above.

11.6.47 Remark As a demonstration of the effectiveness of the sparse methods, the record factorization of RSA768 [22], mentioned below, produced 64 billion linear relations. These were reduced, using structured gaussian elimination, to a system of almost 200 million equations in about that many unknowns. This system was still sparse, with the average equation involving about 150 unknowns. The block Wiedemann method was then used to solve the resulting system.

11.6.11 Current discrete log records

11.6.48 Remark Extreme caution should be exercised when drawing any inferences about relative performance of various integer factorization and discrete log algorithms from the record results listed here. The computing resources, as well as effort involved in programming, varied widely among the various projects.

11.6.49 Remark As of the time of writing (early 2012), the largest cryptographically hard integer (i.e., one that was chosen specifically to resist all known factoring attacks, and is a product of two roughly equal primes) that has been factored is RSA768, a 768-bit (232 decimal digit) integer from the RSA challenge list [22]. This was the result of a large collaboration across the globe stretching over more than two years, and used the general number field sieve.

11.6.50 Remark The largest discrete log case for a prime field F_p (with p chosen to resist simple attacks) that has been solved is for a 530-bit (160 decimal digit) prime p. This was accomplished by T. Kleinjung in 2007 [21]. The number field sieve was used.

11.6.51 Remark In fields of characteristic 2, the largest case that has been solved is that of F_q with $q = 2^{613}$, using the function field sieve. (An earlier record was for $q = 2^{607}$ using the Coppersmith algorithm.) This computation took several weeks on a handful of processors, and was carried out by A. Joux and R. Lercier in 2005 [16].

11.6.52 Remark The largest generic discrete log problem that has been solve in a hard case is that of discrete logs over an elliptic curve modulo a 112-bit prime, thus a group of size about 2^{112}. This is due to J. W. Bos and M. E. Kaihara [3], and

was done in 2009.. Right now, a lage multi-year collaborative effort is under way to break the Certicom ECC2K-130 challenge, which involves computing discrete logs on an elliptic curve over a field with 2^{131} elements [2]. All these efforts rely on parallelized versions of the Pollard rho method.

Bibliography

[1] Leonard M. Adleman, The function field sieve, In *Algorithmic number theory (Ithaca, NY, 1994)*, volume 877 of *Lecture Notes in Comput. Sci.*, pages 108–121. Springer, Berlin, 1994. [67]

[2] Daniel V. Bailey, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, H.-C. Chen, C.-M. Cheng, G. van Damme, G. de Meulenaer, L. J. Dominguez Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A. Van Herrewege, and B.-Y. Yang, Breaking ECC2K-130, preprint, http://eprint.iacr.org/2009/541, 2009. [69]

[3] J. Bos and M. E. Kaihara, Playstation 3 computing breaks 2^{60} barrier: 112-bit prime ECDLP solved, online annoucement, http://lacal.epfl.ch/112bit_prime, 2009. [68]

[4] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval, The group Diffie-Hellman problems, In *Selected areas in cryptography*, volume 2595 of *Lecture Notes in Comput. Sci.*, pages 325–338. Springer, Berlin, 2003. [62]

[5] Johannes A. Buchmann and Christine S. Hollinger, On smooth ideals in number fields, *J. Number Theory*, 59(1):82–87, 1996. [67]

[6] J. H Cheon, J. Hong, and M. Kim, Accelerating pollard's rho algorithm in finite fields, *Journal of Cryptology*, 25(2):185–242, 2012. [65]

[7] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren, editors, *Handbook of elliptic and hyperelliptic curve cryptography*, Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006. [61]

[8] An Commeine and Igor Semaev, An algorithm to solve the discrete logarithm problem with the number field sieve, In *Public key cryptography—PKC 2006*, volume 3958 of *Lecture Notes in Comput. Sci.*, pages 174–190. Springer, Berlin, 2006. [67]

[9] Don Coppersmith, Fast evaluation of logarithms in fields of characteristic two, *IEEE Trans. Inform. Theory*, 30(4):587–594, 1984. [67]

[10] Don Coppersmith, Solving linear equations over GF(2): block Lanczos algorithm, *Linear Algebra Appl.*, 192:33–60, 1993, Computational linear algebra in algebraic and related problems (Essen, 1992). [68]

[11] Don Coppersmith, Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm, *Math. Comp.*, 62(205):333–350, 1994. [68]

[12] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel, Discrete logarithms in GF(p), *Algorithmica*, 1(1):1–15, 1986. [68]

[13] Daniel M. Gordon, Discrete logarithms in GF(p) using the number field sieve, *SIAM J. Discrete Math.*, 6(1):124–138, 1993. [67]

[14] Johan Hstad and Mats Näslund, The security of all RSA and discrete log bits, *J. ACM*, 51(2):187–230, 2004. [62]

[15] Adolf Hildebrand and Gérald Tenenbaum, Integers without large prime factors, *J. Théor. Nombres Bordeaux*, 5(2):411–484, 1993. [67]

[16] A. Joux, Discrete logarithms in GF(2^{607}) and GF(2^{613}), mailing list announcement, https://listserv.nodak.edu/cgi-

bin/wa.exe?A2=ind0509&L=NMBRTHRY&P=R1490&D=0&I=-3&T=0, 2005. [68]

[17] Antoine Joux and Reynald Lercier, The function field sieve is quite special, In *Algorithmic number theory (Sydney, 2002)*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 431–445. Springer, Berlin, 2002. [67]

[18] Antoine Joux and Reynald Lercier, Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method, *Math. Comp.*, 72(242):953–967 (electronic), 2003. [67]

[19] Antoine Joux and Reynald Lercier, The function field sieve in the medium prime case, In *Advances in cryptology—EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Comput. Sci.*, pages 254–270. Springer, Berlin, 2006. [67]

[20] Jeong Han Kim, Ravi Montenegro, Yuval Peres, and Prasad Tetali, A birthday paradox for Markov chains with an optimal bound for collision in the Pollard rho algorithm for discrete logarithm, *Ann. Appl. Probab.*, 20(2):495–521, 2010. [65]

[21] T. Kleinjung, Discrete logarithms in GF(p) – 160 digits, mailing list announcement, https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0702&L=NMBRTHRY&P=R45&D=0&I=-3&T=0, 2007. [68]

[22] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann, Factorization of a 768-bit RSA modulus, In *Advances in cryptology—CRYPTO 2010*, volume 6223 of *Lecture Notes in Comput. Sci.*, pages 333–350. Springer, Berlin, 2010. [68]

[23] Donald E. Knuth, *The art of computer programming. Vol. 2: Seminumerical algorithms*, Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1969. [65]

[24] B. LaMacchia and A. Odlyzko, Solving large sparse linear systems over finite fields, In *Advances in Cryptology-CRYPT090*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer, 1991. [68]

[25] Rudolf Lidl and Harald Niederreiter, *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*, Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1983, With a foreword by P. M. Cohn. [61, 62]

[26] Rudolf Lidl and Harald Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, 1986. [61]

[27] Kevin S. McCurley, Cryptographic key distribution and computation in class groups, In *Number theory and applications (Banff, AB, 1988)*, volume 265 of *NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci.*, pages 459–479. Kluwer Acad. Publ., Dordrecht, 1989. [62]

[28] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997, With a foreword by Ronald L. Rivest. [61]

[29] Peter L. Montgomery, A block Lanczos algorithm for finding dependencies over GF(2), In *Advances in cryptology—EUROCRYPT '95 (Saint-Malo, 1995)*, volume 921 of *Lecture Notes in Comput. Sci.*, pages 106–120. Springer, Berlin, 1995. [68]

[30] Gary L. Mullen and David White, A polynomial representation for logarithms in GF(q), *Acta Arith.*, 47(3):255–261, 1986. [64]

[31] V. I. Nechaev, On the complexity of a deterministic algorithm for a discrete logarithm, *Mat. Zametki*, 55(2):91–101, 189, 1994. [63]

[32] Harald Niederreiter, A short proof for explicit formulas for discrete logarithms in finite fields, *Appl. Algebra Engrg. Comm. Comput.*, 1(1):55–57, 1990. [64]

[33] A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, In *Advances in cryptology (Paris, 1984)*, volume 209 of *Lecture Notes in Comput. Sci.*, pages 224–314. Springer, Berlin, 1985. [61, 66, 67, 68]

[34] Andrew Odlyzko, Discrete logarithms: the past and the future, *Des. Codes Cryptogr.*, 19(2-3):129–145, 2000, Towards a quarter-century of public key cryptography. [61]

[35] Daniel Panario, Xavier Gourdon, and Philippe Flajolet, An analytic approach to smooth polynomials over finite fields, In *Algorithmic number theory (Portland, OR, 1998)*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 226–236. Springer, Berlin, 1998. [67]

[36] Stephen C. Pohlig and Martin E. Hellman, An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE Trans. Information Theory*, IT-24(1):106–110, 1978. [63]

[37] J. M. Pollard, Monte Carlo methods for index computation (mod p), *Math. Comp.*, 32(143):918–924, 1978. [65]

[38] J. M. Pollard, Kangaroos, Monopoly and discrete logarithms, *J. Cryptology*, 13(4):437–447, 2000. [65]

[39] Oliver Schirokauer, The special function field sieve, *SIAM J. Discrete Math.*, 16(1):81–98, 2002. [67]

[40] Oliver Schirokauer, The impact of the number field sieve on the discrete logarithm problem in finite fields, In *Algorithmic number theory: lattices, number fields, curves and cryptography*, volume 44 of *Math. Sci. Res. Inst. Publ.*, pages 397–420. Cambridge Univ. Press, Cambridge, 2008. [67]

[41] Oliver Schirokauer, The number field sieve for integers of low weight, *Math. Comp.*, 79(269):583–602, 2010. [67]

[42] Eira J. Scourfield, On ideals free of large prime factors, *J. Théor. Nombres Bordeaux*, 16(3):733–772, 2004. [67]

[43] Daniel Shanks, Class number, a theory of factorization, and genera, In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)*, pages 415–440. Amer. Math. Soc., Providence, R.I., 1971. [64]

[44] Peter W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comput.*, 26(5):1484–1509, 1997. [63]

[45] Victor Shoup, Lower bounds for discrete logarithms and related problems, In *Advances in cryptology—EUROCRYPT '97 (Konstanz)*, volume 1233 of *Lecture Notes in Comput. Sci.*, pages 256–266. Springer, Berlin, 1997. [63]

[46] Andreas Stein and Edlyn Teske, Optimized baby step–giant step methods, *J. Ramanujan Math. Soc.*, 20(1):27–58, 2005. [65]

[47] Douglas R. Stinson, *Cryptography*, Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, third edition, 2006, Theory and practice. [61]

[48] Edlyn Teske, Square-root algorithms for the discrete logarithm problem (a survey), In *Public-key cryptography and computational number theory (Warsaw, 2000)*, pages 283–301. de Gruyter, Berlin, 2001. [65]

[49] Paul C. van Oorschot and Michael J. Wiener, Parallel collision search with cryptanalytic applications, *J. Cryptology*, 12(1):1–28, 1999. [65]

[50] Zhe-Xian Wan, A shorter proof for an explicit formula for discrete logarithms in finite fields, *Discrete Math.*, 308(21):4914–4915, 2008. [64]

[51] A. Wells Jr, A polynomial form for logarithms modulo a prime (corresp.), *Information Theory, IEEE Transactions on*, 30(6):845–846, 1984. [64]

[52] Douglas H. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Trans. Inform. Theory*, 32(1):54–62, 1986. [68]

Index