15

Subexponential-time discrete logarithms and factoring

This chapter presents subexponential-time algorithms for computing discrete logarithms and for factoring integers. These algorithms share a common technique, which makes essential use of the notion of a **smooth number**.

15.1 Smooth numbers

If y is a non-negative real number and m is a positive integer, then we say that m is **y-smooth** if all prime divisors of m are at most y.

For $0 \leq y \leq x$, let us define $\Psi(y, x)$ to be the number of y-smooth integers up to x. The following theorem gives us a lower bound on $\Psi(y, x)$, which will be crucial in the analysis of our discrete logarithm and factoring algorithms.

Theorem 15.1. *Let y be a function of x such that*

$$\frac{y}{\log x} \to \infty \text{ and } u := \frac{\log x}{\log y} \to \infty$$

as $x \to \infty$. Then

$$\Psi(y, x) \geq x \cdot \exp[(-1 + o(1))u \log \log x].$$

Proof. Let us write $u = \lfloor u \rfloor + \delta$, where $0 \leq \delta < 1$. Let us split the primes up to y into two sets: the set V of "very small" primes that are at most $y^\delta/2$, and the set W of other primes that are greater than $y^\delta/2$ but at most y. To simplify matters, let us also include the integer 1 in the set V.

By Bertrand's postulate (Theorem 5.8), there exists a constant $C > 0$ such that $|W| \geq Cy/\log y$ for sufficiently large y. By the assumption that $y/\log x \to \infty$ as $x \to \infty$, we also have $|W| \geq 2\lfloor u \rfloor$ for sufficiently large x.

To derive the lower bound, we shall count those integers that can be built up by multiplying together $\lfloor u \rfloor$ distinct elements of W, together with one element of V.

These products are clearly distinct, y-smooth numbers, and each is bounded by x, since each is at most $y^{\lfloor u \rfloor} y^{\delta} = y^u = x$.

If S denotes the set of all of these products, then for x sufficiently large, we have

$$\begin{aligned} |S| &= \binom{|W|}{\lfloor u \rfloor} \cdot |V| \\ &= \frac{|W|(|W|-1)\cdots(|W|-\lfloor u \rfloor+1)}{\lfloor u \rfloor !} \cdot |V| \\ &\geq \left(\frac{|W|}{2u}\right)^{\lfloor u \rfloor} \cdot |V| \\ &\geq \left(\frac{Cy}{2u \log y}\right)^{\lfloor u \rfloor} \cdot |V| \\ &= \left(\frac{Cy}{2 \log x}\right)^{u-\delta} \cdot |V|. \end{aligned}$$

Taking logarithms, we have

$$\begin{aligned} \log|S| &\geq (u-\delta)(\log y - \log\log x + \log(C/2)) + \log|V| \\ &= \log x - u \log\log x + (\log|V| - \delta \log y) + \\ &\quad O(u + \log\log x). \end{aligned} \tag{15.1}$$

To prove the theorem, it suffices to show that

$$\log|S| \geq \log x - (1+o(1))u \log\log x.$$

Under our assumption that $u \to \infty$, the term $O(u + \log\log x)$ in (15.1) is clearly $o(u \log\log x)$, and so it will suffice to show that the term $(\log|V| - \delta \log y)$ is also $o(u \log\log x)$. But by Chebyshev's theorem (Theorem 5.1), for some positive constant D, we have

$$Dy^{\delta}/\log y \leq |V| \leq y^{\delta},$$

and taking logarithms, and again using the fact that $u \to \infty$, we have

$$\log|V| - \delta \log y = O(\log\log y) = o(u \log\log x). \quad \square$$

15.2 An algorithm for discrete logarithms

We now present a probabilistic, subexponential-time algorithm for computing discrete logarithms. The input to the algorithm is p, q, γ, α, where p and q are primes, with $q \mid (p-1)$, γ is an element of $\mathbb{Z}_p^*$ generating a subgroup G of $\mathbb{Z}_p^*$ of order q, and $\alpha \in G$.

We shall make the simplifying assumption that $q^2 \nmid (p-1)$, which is equivalent to saying that $q \nmid m := (p-1)/q$. Although not strictly necessary, this assumption simplifies the design and analysis of the algorithm, and moreover, for cryptographic applications, this assumption is almost always satisfied. Exercises 15.1–15.3 below explore how this assumption may be lifted, as well as other generalizations.

At a high level, the main goal of our discrete logarithm algorithm is to find a random representation of 1 with respect to γ and α—as discussed in Exercise 11.12, this allows us to compute $\log_\gamma \alpha$ (with high probability). More precisely, our main goal is to compute integers r and s in a probabilistic fashion, such that $\gamma^r \alpha^s = 1$ and $[s]_q$ is uniformly distributed over $\mathbb{Z}_q$. Having accomplished this, then with probability $1 - 1/q$, we shall have $s \not\equiv 0 \pmod{q}$, which allows us to compute $\log_\gamma \alpha$ as $-rs^{-1} \bmod q$.

Let H be the subgroup of $\mathbb{Z}_p^*$ of order m. Our assumption that $q \nmid m$ implies that $G \cap H = \{1\}$, since the multiplicative order of any element in the intersection must divide both q and m, and so the only possibility is that the multiplicative order is 1. Therefore, the map $\rho : G \times H \to \mathbb{Z}_p^*$ that sends (β, δ) to $\beta\delta$ is injective (Theorem 6.25), and since $|\mathbb{Z}_p^*| = qm$, it must be surjective as well.

We shall use this fact in the following way: if β is chosen uniformly at random from G, and δ is chosen uniformly at random from H (and independent of β), then $\beta\delta$ is uniformly distributed over $\mathbb{Z}_p^*$. Furthermore, since H is the image of the q-power map on $\mathbb{Z}_p^*$, we may generate a random $\delta \in H$ simply by choosing $\hat{\delta} \in \mathbb{Z}_p^*$ at random, and setting $\delta := \hat{\delta}^q$.

The discrete logarithm algorithm uses a "smoothness parameter" y. We will discuss choice of y below, when we analyze the running time of the algorithm; for now, we only assume that $y < p$. Let $p_1, \ldots, p_k$ be an enumeration of the primes up to y. Let $\pi_i := [p_i]_p \in \mathbb{Z}_p^*$ for $i = 1, \ldots, k$.

The algorithm has two stages.

In the first stage, we find relations of the form

$$\gamma^{r_i} \alpha^{s_i} \delta_i = \pi_1^{e_{i1}} \ldots \pi_k^{e_{ik}}, \tag{15.2}$$

for $i = 1, \ldots, k+1$, where $r_i, s_i, e_{i1}, \ldots, e_{ik} \in \mathbb{Z}$ and $\delta_i \in H$ for each i.

We obtain each such relation by a randomized search, as follows: we choose $r_i, s_i \in \{0, \ldots, q-1\}$ at random, as well as $\hat{\delta}_i \in \mathbb{Z}_p^*$ at random; we then compute $\delta_i := \hat{\delta}_i^q$, $\beta_i := \gamma^{r_i}\alpha^{s_i}$, and $m_i := \operatorname{rep}(\beta_i\delta_i)$. Now, the value β_i is uniformly distributed over G, while δ_i is uniformly distributed over H; therefore, the product $\beta_i\delta_i$ is uniformly distributed over $\mathbb{Z}_p^*$, and hence m_i is uniformly distributed over $\{1, \ldots, p-1\}$. Next, we simply try to factor m_i by trial division, trying all the primes $p_1, \ldots, p_k$ up to y. If we are lucky, we completely factor m_i in this way, obtaining a factorization

$$m_i = p_1^{e_{i1}} \cdots p_k^{e_{ik}},$$

for some exponents $e_{i1}, \ldots, e_{ik}$, and we get the relation (15.2). If we are unlucky, then we simply keep trying until we are lucky.

For $i = 1, \ldots, k+1$, let $v_i := (e_{i1}, \ldots, e_{ik}) \in \mathbb{Z}^{\times k}$, and let $\bar{v}_i$ denote the image of v_i in $\mathbb{Z}_q^{\times k}$ (i.e., $\bar{v}_i := ([e_{i1}]_q, \ldots, [e_{ik}]_q)$). Since $\mathbb{Z}_q^{\times k}$ is a vector space over the field $\mathbb{Z}_q$ of dimension k, the family of vectors $\bar{v}_1, \ldots, \bar{v}_{k+1}$ must be linearly dependent. The second stage of the algorithm uses Gaussian elimination over $\mathbb{Z}_q$ (see §14.4) to find a linear dependence among the vectors $\bar{v}_1, \ldots, \bar{v}_{k+1}$, that is, to find integers $c_1, \ldots, c_{k+1} \in \{0, \ldots, q-1\}$, not all zero, such that

$$(e_1, \ldots, e_k) := c_1 v_1 + \cdots + c_{k+1} v_{k+1} \in q\mathbb{Z}^{\times k}.$$

Raising each equation (15.2) to the corresponding power c_i, and multiplying them all together, we obtain

$$\gamma^r \alpha^s \delta = \pi_1^{e_1} \cdots \pi_k^{e_k},$$

where

$$r := \sum_{i=1}^{k+1} c_i r_i, \; s := \sum_{i=1}^{k+1} c_i s_i, \text{ and } \delta := \prod_{i=1}^{k+1} \delta_i^{c_i}.$$

Now, $\delta \in H$, and since each e_j is a multiple of q, we also have $\pi_j^{e_j} \in H$ for $j = 1, \ldots, k$. It follows that $\gamma^r \alpha^s \in H$. But since $\gamma^r \alpha^s \in G$ as well, and $G \cap H = \{1\}$, it follows that $\gamma^r \alpha^s = 1$. If we are lucky (and we will be with overwhelming probability, as we discuss below), we will have $s \not\equiv 0 \pmod{q}$, in which case, we can compute $s' := s^{-1} \bmod q$, obtaining

$$\alpha = \gamma^{-rs'},$$

and hence $-rs' \bmod q$ is the discrete logarithm of α to the base γ. If we are very unlucky, we will have $s \equiv 0 \pmod{q}$, at which point the algorithm simply quits, reporting "failure."

The entire algorithm, called Algorithm SEDL, is presented in Fig. 15.1.

As already argued above, if Algorithm SEDL does not output "failure," then its output is indeed the discrete logarithm of α to the base γ. There remain three questions to answer:

1. What is the expected running time of Algorithm SEDL?
2. How should the smoothness parameter y be chosen so as to minimize the expected running time?
3. What is the probability that Algorithm SEDL outputs "failure"?

Let us address these questions in turn. As for the expected running time, let σ be the probability that a random element of $\{1, \ldots, p-1\}$ is y-smooth. Then

$i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    repeat
        choose $r_i, s_i \in \{0, \dots, q-1\}$ at random
        choose $\hat{\delta}_i \in \mathbb{Z}_p^*$ at random
        $\beta_i \leftarrow \gamma^{r_i}\alpha^{s_i}, \quad \delta_i \leftarrow \hat{\delta}_i^q, \quad m_i \leftarrow \mathrm{rep}(\beta_i\delta_i)$
        test if m_i is y-smooth (trial division)
    until $m_i = p_1^{e_{i1}} \cdots p_k^{e_{ik}}$ for some integers $e_{i1}, \dots, e_{ik}$
until $i = k + 1$

set $v_i \leftarrow (e_{i1}, \dots, e_{ik}) \in \mathbb{Z}^{\times k}$ for $i = 1, \dots, k+1$

apply Gaussian elimination over $\mathbb{Z}_q$ to find integers $c_1, \dots, c_{k+1} \in$
    $\{0, \dots, q-1\}$, not all zero, such that
    $c_1 v_1 + \cdots + c_{k+1} v_{k+1} \in q\mathbb{Z}^{\times k}$.

$r \leftarrow \sum_{i=1}^{k+1} c_i r_i, \quad s \leftarrow \sum_{i=1}^{k+1} c_i s_i$

if $s \equiv 0 \pmod{q}$
    then output "failure"
    else output $-rs^{-1} \bmod q$

Fig. 15.1. Algorithm SEDL

the expected number of attempts needed to produce a single relation is σ^{-1}, and so the expected number of attempts to produce $k+1$ relations is $(k+1)\sigma^{-1}$. In each attempt, we perform trial division using $p_1, \dots, p_k$, along with a few other minor computations, leading to a total expected running time in stage 1 of $k^2\sigma^{-1} \cdot \mathrm{len}(p)^{O(1)}$. The running time in stage 2 is dominated by the Gaussian elimination step, which takes time $k^3 \cdot \mathrm{len}(p)^{O(1)}$. Thus, if Z is the total running time of the algorithm, then we have

$$\mathsf{E}[Z] \le (k^2\sigma^{-1} + k^3) \cdot \mathrm{len}(p)^{O(1)}. \tag{15.3}$$

Let us assume for the moment that

$$y = \exp[(\log p)^{\lambda + o(1)}] \tag{15.4}$$

for some constant λ with $0 < \lambda < 1$. Our final choice of y will indeed satisfy this assumption. Consider the probability σ. We have

$$\sigma = \Psi(y, p-1)/(p-1) = \Psi(y, p)/(p-1) \ge \Psi(y, p)/p,$$

where for the second equality we use the assumption that $y < p$, so p is not y-smooth. With our assumption (15.4), we may apply Theorem 15.1 (with the given value of y and $x := p$), obtaining

$$\sigma \geq \exp[(-1 + o(1))(\log p / \log y) \log\log p].$$

By Chebyshev's theorem (Theorem 5.1), we know that $k = \Theta(y/\log y)$, and so $\log k = (1 + o(1)) \log y$. Moreover, assumption (15.4) implies that the factor $\mathrm{len}(p)^{O(1)}$ in (15.3) is of the form $\exp[o(\min(\log y, \log p/\log y))]$, and so we have

$$\mathsf{E}[Z] \leq \exp[(1 + o(1)) \max\{(\log p/\log y) \log\log p + 2\log y,\ 3\log y\}]. \quad (15.5)$$

Let us find the value of y that minimizes the right-hand side of (15.5), ignoring the "o(1)" terms. Let $\mu := \log y$, $A := \log p \log\log p$, $S_1 := A/\mu + 2\mu$, and $S_2 := 3\mu$. We want to find μ that minimizes $\max\{S_1, S_2\}$. Using a little calculus, one sees that S_1 is minimized at $\mu = (A/2)^{1/2}$. With this choice of μ, we have $S_1 = (2\sqrt{2})A^{1/2}$ and $S_2 = (3/\sqrt{2})A^{1/2} < S_1$. Thus, choosing

$$y = \exp[(1/\sqrt{2})(\log p \log\log p)^{1/2}],$$

we obtain

$$\mathsf{E}[Z] \leq \exp[(2\sqrt{2} + o(1))(\log p \log\log p)^{1/2}].$$

That takes care of the first two questions, although strictly speaking, we have only obtained an upper bound for the expected running time, and we have not shown that the choice of y is actually optimal, but we shall nevertheless content ourselves (for now) with these results. Finally, we deal with the third question, on the probability that the algorithm outputs "failure."

Lemma 15.2. *The probability that Algorithm* SEDL *outputs "failure" is* $1/q$.

Proof. Let $\mathcal{F}$ be the event that the algorithm outputs "failure." For $i = 1, \ldots, k+1$, we may view the *final* values assigned to r_i, s_i, δ_i, and m_i as random variables, which we shall denote by these same names (to avoid additional notation). Similarly, we may view s as a random variable.

Let $m'_1, \ldots, m'_{k+1}$ be arbitrary, fixed y-smooth numbers, and let $\mathcal{B}$ be the event that $m_1 = m'_1, \ldots, m_{k+1} = m'_{k+1}$. We shall show that $\mathsf{P}[\mathcal{F} \mid \mathcal{B}] = 1/q$, and since this holds for all relevant $\mathcal{B}$, it follows by total probability that $\mathsf{P}[\mathcal{F}] = 1/q$.

For the rest of the argument, we focus on the conditional distribution given $\mathcal{B}$. With respect to this conditional distribution, the distribution of each random variable (r_i, s_i, δ_i) is (essentially) the uniform distribution on the set

$$P_i := \{(r', s', \delta') \in I_q \times I_q \times H : \gamma^{r'} \alpha^{s'} \delta' = [m'_i]_p\},$$

where $I_q := \{0, \ldots, q-1\}$; also, the family of random variables $\{(r_i, s_i, \delta_i)\}_{i=1}^{k+1}$

is mutually independent. It is easy to see that for $i = 1, \ldots, k + 1$, and for each $s' \in I_q$, there exist unique values $r' \in I_q$ and $\delta' \in H$ such that $(r', s', \delta') \in P_i$. From this, it easily follows that each s_i is uniformly distributed over I_q, and the family of random variables $\{s_i\}_{i=1}^{k+1}$ is mutually independent. Also, the values $c_1, \ldots, c_{k+1}$ computed by the algorithm are *fixed* (as they are determined by $m'_1, \ldots, m'_{k+1}$), and since $s = c_1 s_1 + \cdots + c_{k+1} s_{k+1}$, and not all the c_i's are zero modulo q, it follows that $s \bmod q$ is uniformly distributed over I_q, and so is equal to zero with probability $1/q$. □

Let us summarize the above discussion in the following theorem.

Theorem 15.3. *With the smoothness parameter set as*

$$y := \exp[(1/\sqrt{2})(\log p \log\log p)^{1/2}],$$

the expected running time of Algorithm SEDL *is at most*

$$\exp[(2\sqrt{2} + o(1))(\log p \log\log p)^{1/2}].$$

The probability that Algorithm SEDL *outputs "failure" is* $1/q$.

In the description and analysis of Algorithm SEDL, we have assumed that the primes $p_1, \ldots, p_k$ were pre-computed. Of course, we can construct this list of primes using, for example, the sieve of Eratosthenes (see §5.4), and the running time of this pre-computation will be dominated by the running time of Algorithm SEDL.

In the analysis of Algorithm SEDL, we relied crucially on the fact that in generating a relation, each candidate element $\gamma^{r_i}\alpha^{s_i}\delta_i$ was uniformly distributed over $\mathbb{Z}_p^*$. If we simply left out the δ_i's, then the candidate element would be uniformly distributed over the subgroup G, and Theorem 15.1 simply would not apply. Although the algorithm might anyway work as expected, we would not be able to prove this.

EXERCISE 15.1. Using the result of Exercise 14.19, show how to modify Algorithm SEDL to work in the case where $p - 1 = q^e m$, $e > 1$, $q \nmid m$, γ generates the subgroup G of $\mathbb{Z}_p^*$ of order q^e, and $\alpha \in G$. Your algorithm should compute $\log_\gamma \alpha$ with roughly the same expected running time and success probability as Algorithm SEDL.

EXERCISE 15.2. Using the algorithm of the previous exercise as a subroutine, design and analyze an algorithm for the following problem. The input is p, q, γ, α, where p is a prime, q is a prime dividing $p - 1$, γ generates the subgroup G of $\mathbb{Z}_p^*$ of order q, and $\alpha \in G$; note that we may have $q^2 \mid (p - 1)$. The output is $\log_\gamma \alpha$. Your algorithm should always succeed in computing this discrete logarithm, and its expected running time should be bounded by a constant times the expected running time of the algorithm of the previous exercise.

EXERCISE 15.3. Using the result of Exercise 14.20, show how to modify Algorithm SEDL to solve the following problem: given a prime p, a generator γ for $\mathbb{Z}_p^*$, and an element $\alpha \in \mathbb{Z}_p^*$, compute $\log_\gamma \alpha$. Your algorithm should work without knowledge of the factorization of $p-1$; its expected running time should be roughly the same as that of Algorithm SEDL, but its success probability may be lower. In addition, explain how the success probability may be significantly increased at almost no cost by collecting a few extra relations.

EXERCISE 15.4. Let $n = pq$, where p and q are distinct, large primes. Let e be a prime, with $e < n$ and $e \nmid (p-1)(q-1)$. Let x be a positive integer, with $x < n$. Suppose you are given n (but not its factorization!) along with e and x. In addition, you are given access to two "oracles," which you may invoke as often as you like.

- The first oracle is a "challenge oracle": each invocation of the oracle produces a "challenge" $a \in \{1, \ldots, x\}$—distributed uniformly, and independent of all other challenges.
- The second oracle is a "solution oracle": you invoke this oracle with the index of a previous challenge oracle; if the corresponding challenge was a, the solution oracle returns the eth root of a modulo n; that is, the solution oracle returns $b \in \{1, \ldots, n-1\}$ such that $b^e \equiv a \pmod{n}$—note that b always exists and is uniquely determined.

Let us say that you "win" if you are able to compute the eth root modulo n of any challenge, but *without* invoking the solution oracle with the corresponding index of the challenge (otherwise, winning would be trivial, of course).

(a) Design a probabilistic algorithm that wins the above game, using an expected number of

$$\exp[(c + o(1))(\log x \log\log x)^{1/2}] \cdot \operatorname{len}(n)^{O(1)}$$

steps, for some constant c, where a "step" is either a computation step or an oracle invocation (either challenge or solution). Hint: Gaussian elimination over the field $\mathbb{Z}_e$.

(b) Suppose invocations of the challenge oracle are "cheap," while invocations of the solution oracle are relatively "expensive." How would you modify your strategy in part (a)?

Exercise 15.4 has implications in cryptography. A popular way of implementing a public-key primitive known as a "digital signature" works as follows: to digitally sign a message M (which may be an arbitrarily long bit string), first apply a "hash function" or "message digest" H to M, obtaining an integer a in some fixed range $\{1, \ldots, x\}$, and then compute the signature of M as the eth root b of a modulo n. Anyone can verify that such a signature b is correct by checking that $b^e \equiv H(M) \pmod{n}$; however, it would appear to be difficult to "forge" a signature without knowing the factorization of n. Indeed, one can prove the security of this signature scheme by assuming that it is hard to compute the eth root of a random number modulo n, and by making the heuristic assumption that H is a random function (see §15.5). However, for this proof to work, the value of x must be close to n; otherwise, if x is significantly smaller than n, as the result of this exercise, one can break the signature scheme at a cost that is roughly the same as the cost of factoring numbers around the size of x, rather than the size of n.

15.3 An algorithm for factoring integers

We now present a probabilistic, subexponential-time algorithm for factoring integers. The algorithm uses techniques very similar to those used in Algorithm SEDL in §15.2.

Let $n > 1$ be the integer we want to factor. We make a few simplifying assumptions. First, we assume that n is odd—this is not a real restriction, since we can always pull out any factors of 2 in a pre-processing step. Second, we assume that n is not a perfect power, that is, not of the form a^b for integers $a > 1$ and $b > 1$—this is also not a real restriction, since we can always partially factor n using the algorithm from Exercise 3.31 if n is a perfect power. Third, we assume that n is not prime—this may be efficiently checked using, say, the Miller–Rabin test (see §10.2). Fourth, we assume that n is not divisible by any primes up to a "smoothness parameter" y—we can ensure this using trial division, and it will be clear that the running time of this pre-computation is dominated by that of the algorithm itself.

With these assumptions, the prime factorization of n is of the form

$$n = q_1^{f_1} \cdots q_w^{f_w},$$

where $w > 1$, the q_i's are distinct, odd primes, each greater than y, and the f_i's are positive integers.

The main goal of our factoring algorithm is to find a random square root of 1 in $\mathbb{Z}_n^*$. Let

$$\begin{aligned} \theta : \quad \mathbb{Z}_n &\to \mathbb{Z}_{q_1^{f_1}} \times \cdots \times \mathbb{Z}_{q_w^{f_w}} \\ [a]_n &\mapsto ([a]_{q_1^{f_1}}, \ldots, [a]_{q_w^{f_w}}) \end{aligned}$$

be the ring isomorphism of the Chinese remainder theorem. The square roots of 1 in $\mathbb{Z}_n^*$ are precisely those elements $\gamma \in \mathbb{Z}_n^*$ such that $\theta(\gamma) = (\pm 1, \ldots, \pm 1)$. If γ is a random square root of 1, then with probability $1 - 2^{-w+1} \geq 1/2$, we have

$\theta(\gamma) = (\gamma_1, \ldots, \gamma_w)$, where the γ_i's are neither all 1 nor all -1 (i.e., $\gamma \neq \pm 1$). If this happens, then $\theta(\gamma - 1) = (\gamma_1 - 1, \ldots, \gamma_w - 1)$, and so we see that some, but not all, of the values $\gamma_i - 1$ will be zero. The value of $\gcd(\mathrm{rep}(\gamma - 1), n)$ is precisely the product of the prime powers $q_i^{f_i}$ such that $\gamma_i - 1 = 0$, and hence this gcd will yield a non-trivial factorization of n, unless $\gamma = \pm 1$.

Let $p_1, \ldots, p_k$ be the primes up to the smoothness parameter y mentioned above. Let $\pi_i := [p_i]_n \in \mathbb{Z}_n^*$ for $i = 1, \ldots, k$.

We first describe a simplified version of the algorithm, after which we modify the algorithm slightly to deal with a technical problem. Like Algorithm SEDL, this algorithm proceeds in two stages. In the first stage, we find relations of the form

$$\alpha_i^2 = \pi_1^{e_{i1}} \cdots \pi_k^{e_{ik}}, \tag{15.6}$$

for $i = 1, \ldots, k + 1$, where $e_{i1}, \ldots, e_{ik} \in \mathbb{Z}$ and $\alpha_i \in \mathbb{Z}_n^*$ for each i.

We can obtain each such relation by randomized search, as follows: we select $\alpha_i \in \mathbb{Z}_n^*$ at random, square it, and try to factor $m_i := \mathrm{rep}(\alpha_i^2)$ by trial division, trying all the primes $p_1, \ldots, p_k$ up to y. If we are lucky, we obtain a factorization

$$m_i = p_1^{e_{i1}} \cdots p_k^{e_{ik}},$$

for some exponents $e_{i1}, \ldots, e_{ik}$, yielding the relation (15.6); if not, we just keep trying.

For $i = 1, \ldots, k + 1$, let $v_i := (e_{i1}, \ldots, e_{ik}) \in \mathbb{Z}^{\times k}$, and let $\bar{v}_i$ denote the image of v_i in $\mathbb{Z}_2^{\times k}$ (i.e., $\bar{v}_i := ([e_{i1}]_2, \ldots, [e_{ik}]_2)$). Since $\mathbb{Z}_2^{\times k}$ is a vector space over the field $\mathbb{Z}_2$ of dimension k, the family of vectors $\bar{v}_1, \ldots, \bar{v}_{k+1}$ must be linearly dependent. The second stage of the algorithm uses Gaussian elimination over $\mathbb{Z}_2$ to find a linear dependence among the vectors $\bar{v}_1, \ldots, \bar{v}_{k+1}$, that is, to find integers $c_1, \ldots, c_{k+1} \in \{0, 1\}$, not all zero, such that

$$(e_1, \ldots, e_k) := c_1 v_1 + \cdots + c_{k+1} v_{k+1} \in 2\mathbb{Z}^{\times k}.$$

Raising each equation (15.6) to the corresponding power c_i, and multiplying them all together, we obtain

$$\alpha^2 = \pi_1^{e_1} \cdots \pi_k^{e_k},$$

where

$$\alpha := \prod_{i=1}^{k+1} \alpha_i^{c_i}.$$

Since each e_i is even, we can compute

$$\beta := \pi_1^{e_1/2} \cdots \pi_k^{e_k/2},$$

and we see that $\alpha^2 = \beta^2$, and hence $(\alpha/\beta)^2 = 1$. Thus, $\gamma := \alpha/\beta$ is a square root of 1 in $\mathbb{Z}_n^*$. A more careful analysis (see below) shows that in fact, γ is uniformly distributed over all square roots of 1, and hence, with probability at least 1/2, if we compute $\gcd(\text{rep}(\gamma - 1), n)$, we get a non-trivial factor of n.

That is the basic idea of the algorithm. There is, however, a technical problem. Namely, in the method outlined above for generating a relation, we attempt to factor $m_i := \text{rep}(\alpha_i^2)$. Thus, the running time of the algorithm will depend in a crucial way on the probability that a random square modulo n is y-smooth. Unfortunately for us, Theorem 15.1 does not say anything about this situation—it only applies to the situation where a number is chosen at random from an interval $[1, x]$. There are (at least) three different ways to address this problem:

1. Ignore it, and just assume that the bounds in Theorem 15.1 apply to random squares modulo n (taking $x := n$ in the theorem).
2. Prove a version of Theorem 15.1 that applies to random squares modulo n.
3. Modify the factoring algorithm, so that Theorem 15.1 applies.

The first choice, while not unreasonable from a practical point of view, is not very satisfying mathematically. It turns out that the second choice is indeed a viable option (i.e., the theorem is true and is not so difficult to prove), but we opt for the third choice, as it is somewhat easier to carry out, and illustrates a probabilistic technique that is more generally useful.

So here is how we modify the basic algorithm. Instead of generating relations of the form (15.6), we generate relations of the form

$$\alpha_i^2 \delta = \pi_1^{e_{i1}} \cdots \pi_k^{e_{ik}}, \tag{15.7}$$

for $i = 1, \ldots, k+2$, where $e_{i1}, \ldots, e_{ik} \in \mathbb{Z}$ and $\alpha_i \in \mathbb{Z}_n^*$ for each i, and $\delta \in \mathbb{Z}_n^*$. Note that the value δ is the same in all relations.

We generate these relations as follows. For the very first relation (i.e., $i = 1$), we repeatedly choose α_1 and δ in $\mathbb{Z}_n^*$ at random, until $\text{rep}(\alpha_1^2\delta)$ is y-smooth. Then, after having found the first relation, we find each subsequent relation (i.e., for $i > 1$) by repeatedly choosing α_i in $\mathbb{Z}_n^*$ at random until $\text{rep}(\alpha_i^2\delta)$ is y-smooth, where δ is the same value that was used in the first relation. Now, Theorem 15.1 will apply directly to determine the success probability of each attempt to generate the first relation. When we have found this relation, the value $\alpha_1^2\delta$ will be uniformly distributed over all y-smooth elements of $\mathbb{Z}_n^*$ (i.e., elements whose integer representations are y-smooth). Consider the various cosets of $(\mathbb{Z}_n^*)^2$ in $\mathbb{Z}_n^*$. Intuitively, it is much more likely that a random y-smooth element of $\mathbb{Z}_n^*$ lies in a coset that contains many y-smooth elements than in a coset with very few, and indeed, it is reasonably likely that the fraction of y-smooth elements in the coset containing δ is not much less than the overall fraction of y-smooth elements in $\mathbb{Z}_n^*$. Therefore, for $i > 1$, each attempt to find a relation should succeed with reasonably high probability. This intuitive argument will be made rigorous in the analysis to follow.

The second stage is then modified as follows. For $i = 1, \ldots, k+2$, let $v_i := (e_{i1}, \ldots, e_{ik}, 1) \in \mathbb{Z}^{\times(k+1)}$, and let $\bar{v}_i$ denote the image of v_i in $\mathbb{Z}_2^{\times(k+1)}$. Since $\mathbb{Z}_2^{\times(k+1)}$ is a vector space over the field $\mathbb{Z}_2$ of dimension $k+1$, the family of vectors $\bar{v}_1, \ldots, \bar{v}_{k+2}$ must be linearly dependent. Therefore, we use Gaussian elimination over $\mathbb{Z}_2$ to find a linear dependence among the vectors $\bar{v}_1, \ldots, \bar{v}_{k+2}$, that is, to find integers $c_1, \ldots, c_{k+2} \in \{0, 1\}$, not all zero, such that

$$(e_1, \ldots, e_{k+1}) := c_1 v_1 + \cdots + c_{k+2} v_{k+2} \in 2\mathbb{Z}^{\times(k+1)}.$$

Raising each equation (15.7) to the corresponding power c_i, and multiplying them all together, we obtain

$$\alpha^2 \delta^{e_{k+1}} = \pi_1^{e_1} \cdots \pi_k^{e_k},$$

where

$$\alpha := \prod_{i=1}^{k+2} \alpha_i^{c_i}.$$

Since each e_i is even, we can compute

$$\beta := \pi_1^{e_1/2} \cdots \pi_k^{e_k/2} \delta^{-e_{k+1}/2},$$

so that $\alpha^2 = \beta^2$ and $\gamma := \alpha/\beta$ is a square root of 1 in $\mathbb{Z}_n^*$.

The entire algorithm, called Algorithm SEF, is presented in Fig. 15.2.

Now the analysis. From the discussion above, it is clear that Algorithm SEF either outputs "failure," or outputs a non-trivial factor of n. So we have the same three questions to answer as we did in the analysis of Algorithm SEDL:

1. What is the expected running time of Algorithm SEF?
2. How should the smoothness parameter y be chosen so as to minimize the expected running time?
3. What is the probability that Algorithm SEF outputs "failure"?

To answer the first question, let σ denote the probability that (the canonical representative of) a random element of $\mathbb{Z}_n^*$ is y-smooth. For $i = 1, \ldots, k+2$, let L_i denote the number of iterations of the inner loop in the ith iteration of the main loop in stage 1; that is, L_i is the number of attempts made in finding the ith relation.

Lemma 15.4. *For $i = 1, \ldots, k+2$, we have $\mathsf{E}[L_i] \le \sigma^{-1}$.*

Proof. We first compute $\mathsf{E}[L_1]$. As δ is chosen uniformly from $\mathbb{Z}_n^*$ and independent of α_1, at each attempt to find a relation, $\alpha_1^2\delta$ is uniformly distributed over $\mathbb{Z}_n^*$,

$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  repeat
    choose $\alpha_i \in \mathbb{Z}_n^*$ at random
    if $i = 1$ then choose $\delta \in \mathbb{Z}_n^*$ at random
    $m_i \leftarrow \operatorname{rep}(\alpha_i^2 \delta)$
    test if m_i is y-smooth (trial division)
  until $m_i = p_1^{e_{i1}} \cdots p_k^{e_{ik}}$ for some integers $e_{i1}, \ldots, e_{ik}$
until $i = k + 2$
set $v_i \leftarrow (e_{i1}, \ldots, e_{ik}, 1) \in \mathbb{Z}^{\times(k+1)}$ for $i = 1, \ldots, k + 2$
apply Gaussian elimination over $\mathbb{Z}_2$ to find integers $c_1, \ldots, c_{k+2} \in \{0, 1\}$, not all zero, such that
  $(e_1, \ldots, e_{k+1}) := c_1 v_1 + \cdots + c_{k+2} v_{k+2} \in 2\mathbb{Z}^{\times(k+1)}$.
$\alpha \leftarrow \prod_{i=1}^{k+2} \alpha_i^{c_i}$, $\beta \leftarrow \pi_1^{e_1/2} \cdots \pi_k^{e_k/2} \delta^{-e_{k+1}/2}$, $\gamma \leftarrow \alpha/\beta$
if $\gamma = \pm 1$
  then output "failure"
  else output $\gcd(\operatorname{rep}(\gamma - 1), n)$

Fig. 15.2. Algorithm SEF

and hence the probability that the attempt succeeds is precisely σ. This means $\mathsf{E}[L_1] = \sigma^{-1}$.

We next compute $\mathsf{E}[L_i]$ for $i > 1$. To this end, let us denote the cosets of $(\mathbb{Z}_n^*)^2$ by $\mathbb{Z}_n^*$ as $C_1, \ldots, C_t$. As it happens, $t = 2^w$, but this fact plays no role in the analysis. For $j = 1, \ldots, t$, let σ_j denote the probability that a random element of C_j is y-smooth, and let τ_j denote the probability that the final value of δ belongs to C_j.

We claim that for $j = 1, \ldots, t$, we have $\tau_j = \sigma_j \sigma^{-1} t^{-1}$. To see this, note that each coset C_j has the same number of elements, namely, $|\mathbb{Z}_n^*| t^{-1}$, and so the number of y-smooth elements in C_j is equal to $\sigma_j |\mathbb{Z}_n^*| t^{-1}$. Moreover, the final value of $\alpha_1^2 \delta$ is equally likely to be any one of the y-smooth numbers in $\mathbb{Z}_n^*$, of which there are $\sigma |\mathbb{Z}_n^*|$, and hence

$$\tau_j = \frac{\sigma_j |\mathbb{Z}_n^*| t^{-1}}{\sigma |\mathbb{Z}_n^*|} = \sigma_j \sigma^{-1} t^{-1},$$

which proves the claim.

Now, for a fixed value of δ and a random choice of $\alpha_i \in \mathbb{Z}_n^*$, one sees that $\alpha_i^2\delta$ is uniformly distributed over the coset containing δ. Therefore, for $j = 1, \ldots, t$, if $\tau_j > 0$, we have

$$\mathsf{E}[L_i \mid \delta \in C_j] = \sigma_j^{-1}.$$

Summing over all $j = 1, \ldots, t$ with $\tau_j > 0$, it follows that

$$\begin{aligned}\mathsf{E}[L_i] &= \sum_{\tau_j > 0} \mathsf{E}[L_i \mid \delta \in C_j] \cdot \mathsf{P}[\delta \in C_j] \\ &= \sum_{\tau_j > 0} \sigma_j^{-1} \cdot \tau_j = \sum_{\tau_j > 0} \sigma_j^{-1} \cdot \sigma_j \sigma^{-1} t^{-1} \leq \sigma^{-1},\end{aligned}$$

which proves the lemma. □

So in stage 1, the expected number of attempts made in generating a single relation is σ^{-1}, each such attempt takes time $k \cdot \mathrm{len}(n)^{O(1)}$, and we have to generate $k+2$ relations, leading to a total expected running time in stage 1 of $\sigma^{-1}k^2 \cdot \mathrm{len}(n)^{O(1)}$. Stage 2 is dominated by the cost of performing Gaussian elimination, which takes time $k^3 \cdot \mathrm{len}(n)^{O(1)}$. Thus, if Z is the total running time of the algorithm, we have

$$\mathsf{E}[Z] \leq (\sigma^{-1}k^2 + k^3) \cdot \mathrm{len}(n)^{O(1)}.$$

By our assumption that n is not divisible by any primes up to y, all y-smooth integers up to $n-1$ are in fact relatively prime to n. Therefore, the number of y-smooth elements of $\mathbb{Z}_n^*$ is equal to $\Psi(y, n-1)$, and since n itself is not y-smooth, this is equal to $\Psi(y, n)$. From this, it follows that

$$\sigma = \Psi(y, n)/|\mathbb{Z}_n^*| \geq \Psi(y, n)/n.$$

The rest of the running time analysis is essentially the same as in the analysis of Algorithm SEDL; that is, assuming $y = \exp[(\log n)^{\lambda + o(1)}]$ for some constant $0 < \lambda < 1$, we obtain

$$\mathsf{E}[Z] \leq \exp[(1 + o(1)) \max\{(\log n/\log y) \log\log n + 2\log y,\ 3\log y\}]. \quad (15.8)$$

Setting $y = \exp[(1/\sqrt{2})(\log n \log\log n)^{1/2}]$, we obtain

$$\mathsf{E}[Z] \leq \exp[(2\sqrt{2} + o(1))(\log n \log\log n)^{1/2}].$$

That basically takes care of the first two questions. As for the third, we have:

Lemma 15.5. *Algorithm* SEF *outputs "failure" with probability* $2^{-w+1} \leq 1/2$.

Proof. Let $\mathcal{F}$ be the event that the algorithm outputs "failure." We may view the *final* values assigned to δ and $\alpha_1, \ldots, \alpha_{k+2}$ as random variables, which we shall denote by these same names. Let $\delta' \in \mathbb{Z}_n^*$ and $\alpha'_1, \ldots, \alpha'_{k+2} \in (\mathbb{Z}_n^*)^2$ be arbitrary, fixed values such that $\mathrm{rep}(\alpha_i'\delta')$ is y-smooth for $i = 1, \ldots, k+2$. Let $\mathcal{B}$ be the event that $\delta = \delta'$ and $\alpha_i^2 = \alpha_i'$ for $i = 1, \ldots, k+2$. We shall show that $\mathsf{P}[\mathcal{F} \mid \mathcal{B}] = 2^{-w+1}$, and since this holds for all relevant $\mathcal{B}$, it follows by total probability that $\mathsf{P}[\mathcal{F}] = 2^{-w+1}$.

For the rest of the argument, we focus on the conditional distribution given $\mathcal{B}$. With respect to this conditional distribution, the distribution of each random variable α_i is (essentially) the uniform distribution on $\rho^{-1}(\{\alpha_i'\})$, where ρ is the squaring map on $\mathbb{Z}_n^*$. Moreover, the family of random variables $\{\alpha_i\}_{i=1}^{k+2}$ is mutually independent. Also, the values β and $c_1, \ldots, c_{k+2}$ computed by the algorithm are *fixed*. It follows (see Exercise 8.14) that the distribution of α is (essentially) the uniform distribution on $\rho^{-1}(\{\beta^2\})$, and hence $\gamma := \alpha/\beta$ is a random square root of 1 in $\mathbb{Z}_n^*$. Thus, $\gamma = \pm 1$ with probability 2^{-w+1}. $\square$

Let us summarize the above discussion in the following theorem.

Theorem 15.6. *With the smoothness parameter set as*

$$y := \exp[(1/\sqrt{2})(\log n \log\log n)^{1/2}],$$

the expected running time of Algorithm SEF *is at most*

$$\exp[(2\sqrt{2} + o(1))(\log n \log\log n)^{1/2}].$$

The probability that Algorithm SEF *outputs "failure" is at most* $1/2$.

EXERCISE 15.5. It is perhaps a bit depressing that after all that work, Algorithm SEF only succeeds (in the worst case) with probability $1/2$. Of course, to reduce the failure probability, we can simply repeat the entire computation—with ℓ repetitions, the failure probability drops to $2^{-\ell}$. However, there is a better way to reduce the failure probability. Suppose that in stage 1, instead of collecting $k+2$ relations, we collect $k+1+\ell$ relations, where $\ell \geq 1$ is an integer parameter.

(a) Show that in stage 2, we can use Gaussian elimination over $\mathbb{Z}_2$ to find integer vectors

$$c^{(j)} = (c_1^{(j)}, \ldots, c_{k+1+\ell}^{(j)}) \in \{0,1\}^{\times(k+1+\ell)} \quad (j = 1, \ldots, \ell)$$

such that

- over the field $\mathbb{Z}_2$, the images of the vectors $c^{(1)}, \ldots, c^{(\ell)}$ in $\mathbb{Z}_2^{\times(k+1+\ell)}$ form a linearly independent family of vectors, and
- for $j = 1, \ldots, \ell$, we have

$$c_1^{(j)} v_1 + \cdots + c_{k+1+\ell}^{(j)} v_{k+1+\ell} \in 2\mathbb{Z}^{\times(k+2)}.$$

(b) Show that given vectors $c^{(1)}, \ldots, c^{(\ell)}$ as in part (a), if for $j = 1, \ldots, \ell$, we set

$$(e_1^{(j)}, \ldots, e_{k+1}^{(j)}) \leftarrow c_1^{(j)} v_1 + \cdots + c_{k+1+\ell}^{(j)} v_{k+1+\ell},$$

$$\alpha^{(j)} \leftarrow \prod_{i=1}^{k+1+\ell} \alpha_i^{c_i^{(j)}},\ \beta^{(j)} \leftarrow \pi_1^{e_1^{(j)}/2} \cdots \pi_k^{e_k^{(j)}/2} \delta^{-e_{k+1}^{(j)}/2},\ \gamma^{(j)} \leftarrow \alpha^{(j)}/\beta^{(j)},$$

then the family of random variables $\gamma^{(1)}, \ldots, \gamma^{(\ell)}$ is mutually independent, with each $\gamma^{(j)}$ uniformly distributed over the set of all square roots of 1 in $\mathbb{Z}_n^*$, and hence at least one of $\gcd(\text{rep}(\gamma^{(j)} - 1), n)$ splits n with probability at least $1 - 2^{-\ell}$.

So, for example, if we set $\ell = 20$, then the failure probability is reduced to less than one in a million, while the increase in running time over Algorithm SEF will hardly be noticeable.

15.4 Practical improvements

Our presentation and analysis of algorithms for discrete logarithms and factoring were geared towards simplicity and mathematical rigor. However, if one really wants to compute discrete logarithms or factor numbers, then a number of important practical improvements should be considered. In this section, we briefly sketch some of these improvements, focusing our attention on algorithms for factoring numbers (although some of the techniques apply to discrete logarithms as well).

15.4.1 Better smoothness density estimates

From an algorithmic point of view, the simplest way to improve the running times of both Algorithms SEDL and SEF is to use a more accurate smoothness density estimate, which dictates a different choice of the smoothness bound y in those algorithms, speeding them up significantly. While our Theorem 15.1 is a valid *lower bound* on the density of smooth numbers, it is not "tight," in the sense that the actual density of smooth numbers is somewhat higher. We quote from the literature the following result:

Theorem 15.7. *Let y be a function of x such that for some $\varepsilon > 0$, we have*

$$y = \Omega((\log x)^{1+\varepsilon}) \text{ and } u := \frac{\log x}{\log y} \to \infty$$

as $x \to \infty$. Then

$$\Psi(y, x) = x \cdot \exp[(-1 + o(1)) u \log u].$$

Proof. See §15.5. □

Let us apply this result to the analysis of Algorithm SEF. Assume that

$$y = \exp[(\log n)^{1/2+o(1)}].$$

Our choice of y will in fact be of this form. With this assumption, we have $\log\log y = (1/2 + o(1))\log\log n$, and using Theorem 15.7, we can improve the inequality (15.8), obtaining instead (as the reader may verify)

$$\mathsf{E}[Z] \le \exp[(1+o(1))\max\{\tfrac{1}{2}(\log n/\log y)\log\log n + 2\log y, 3\log y\}].$$

From this, if we set

$$y := \exp[\tfrac{1}{2}(\log n \log\log n)^{1/2})],$$

we obtain

$$\mathsf{E}[Z] \le \exp[(2+o(1))(\log n \log\log n)^{1/2}].$$

An analogous improvement can be obtained for Algorithm SEDL.

Although this improvement only reduces the constant $2\sqrt{2} \approx 2.828$ to 2, the constant is in the exponent, and so this improvement is not to be scoffed at!

15.4.2 The quadratic sieve algorithm

We now describe a practical improvement to Algorithm SEF. This algorithm, known as the **quadratic sieve**, is faster in practice than Algorithm SEF; however, its analysis is somewhat heuristic.

First, let us return to the simplified version of Algorithm SEF, where we collect relations of the form (15.6). Furthermore, instead of choosing the values α_i at random, we will choose them in a special way, as we now describe. Let

$$\tilde{n} := \lfloor\sqrt{n}\rfloor,$$

and define the polynomial

$$F := (X + \tilde{n})^2 - n \in \mathbb{Z}[X].$$

In addition to the usual "smoothness parameter" y, we need a "sieving parameter" z, whose choice will be discussed below. We shall assume that both y and z are of the form $\exp[(\log n)^{1/2+o(1)}]$, and our ultimate choices of y and z will indeed satisfy this assumption.

For all $s = 1, 2, \ldots, \lfloor z\rfloor$, we shall determine which values of s are "good," in the sense that the corresponding value $F(s)$ is y-smooth. For each good s, since we have $F(s) \equiv (s + \tilde{n})^2 \pmod{n}$, we obtain one relation of the form (15.6), with $\alpha_i := [s + \tilde{n}]_n$. If we find at least $k + 1$ good values of s, then we can apply

Gaussian elimination as usual to find a square root γ of 1 in $\mathbb{Z}_n^*$. Hopefully, we will have $\gamma \neq \pm 1$, allowing us to split n.

Observe that for $1 \leq s \leq z$, we have

$$1 \leq F(s) \leq z^2 + 2zn^{1/2} \leq n^{1/2+o(1)}.$$

Now, although the values $F(s)$ are not at all random, we might expect heuristically that the number of good s up to z is roughly equal to $\hat{\sigma}z$, where $\hat{\sigma}$ is the probability that a random integer in the interval $[1, n^{1/2}]$ is y-smooth, and by Theorem 15.7, we have

$$\hat{\sigma} = \exp[(-\tfrac{1}{4} + o(1))(\log n / \log y) \log\log n].$$

If our heuristics are valid, this already yields an improvement over Algorithm SEF, since now we are looking for y-smooth numbers near $n^{1/2}$, which are much more common than y-smooth numbers near n. But there is another improvement possible; namely, instead of testing each individual number $F(s)$ for smoothness using trial division, we can test them all at once using the following "sieving procedure."

The sieving procedure works as follows. First, we create an array $v[1 \ldots \lfloor z \rfloor]$, and initialize $v[s]$ to $F(s)$, for $1 \leq s \leq z$. Then, for each prime p up to y, we do the following:

1. Compute the roots of the polynomial F modulo p.

   This can be done quite efficiently, as follows. For $p = 2$, F has exactly one root modulo p, which is determined by the parity of $\tilde{n}$. For $p > 2$, we may use the familiar quadratic formula together with an algorithm for computing square roots modulo p, as discussed in Exercise 12.7. A quick calculation shows that the discriminant of F is $4n$, and thus, F has a root modulo p if and only if n is a quadratic residue modulo p, in which case it will have two roots (under our usual assumptions, we cannot have $p \mid n$).

2. Assume that F has v_p distinct roots modulo p lying in the interval $[1, p]$; call them $r_1, \ldots, r_{v_p}$.

   Note that $v_p = 1$ for $p = 2$ and $v_p \in \{0, 2\}$ for $p > 2$. Also note that $F(s) \equiv 0 \pmod{p}$ if and only if $s \equiv r_i \pmod{p}$ for some $i = 1, \ldots, v_p$.

   For $i = 1, \ldots, v_p$, do the following:

   $s \leftarrow r_i$
   while $s \leq z$ do
       repeat $\;v[s] \leftarrow v[s]/p\;$ until $p \nmid v[s]$
       $s \leftarrow s + p$

At the end of this sieving procedure, the good values of s may be identified as precisely those such that $v[s] = 1$. The running time of this sieving procedure is at most $\mathrm{len}(n)^{O(1)}$ times

$$\sum_{p \le y} \frac{z}{p} = z \sum_{p \le y} \frac{1}{p} = O(z \log\log y) = z^{1+o(1)}.$$

Here, we have made use of Theorem 5.10, although this is not really necessary—for our purposes, the bound $\sum_{p \le y} 1/p = O(\log y)$ would suffice. Note that this sieving procedure is a factor of $k^{1+o(1)}$ faster than the method for finding smooth numbers based on trial division. With just a little extra book-keeping, we can not only identify the good values of s but also compute the factorization of $F(s)$ into primes, at essentially no extra cost.

Now, let us put together all the pieces. We have to choose z just large enough so as to find at least $k + 1$ good values of s up to z. So we should choose z so that $z \approx k/\hat{\sigma}$—in practice, we could choose an initial estimate for z, and if this choice of z does not yield enough relations, we could keep doubling z until we do get enough relations. Assuming that $z \approx k/\hat{\sigma}$, the cost of sieving is $(k/\hat{\sigma})^{1+o(1)}$, or

$$\exp[(1 + o(1))(\tfrac{1}{4}(\log n/\log y)\log\log n + \log y)].$$

The cost of Gaussian elimination is still $O(k^3)$, or

$$\exp[(3 + o(1))\log y].$$

Thus, the total running time is bounded by

$$\exp[(1 + o(1))\max\{\tfrac{1}{4}(\log n/\log y)\log\log n + \log y,\ 3\log y\}].$$

Let $\mu := \log y$, $A := (1/4)\log n \log\log n$, $S_1 := A/\mu + \mu$ and $S_2 := 3\mu$, and let us find the value of μ that minimizes $\max\{S_1, S_2\}$. Using a little calculus, one finds that S_1 is minimized at $\mu = A^{1/2}$. For this value of μ, we have $S_1 = 2A^{1/2}$ and $S_2 = 3A^{1/2} > S_1$, and so this choice of μ is a bit larger than optimal. For $\mu < A^{1/2}$, S_1 is decreasing (as a function of μ), while S_2 is always increasing. It follows that the optimal value of μ is obtained by setting

$$A/\mu + \mu = 3\mu,$$

and solving for μ. This yields $\mu = (A/2)^{1/2}$. So setting

$$y := \exp[(1/2\sqrt{2})(\log n \log\log n)^{1/2}],$$

the total running time of the quadratic sieve factoring algorithm is bounded by

$$\exp[(3/2\sqrt{2} + o(1))(\log n \log\log n)^{1/2}].$$

Thus, we have reduced the constant in the exponent from 2 (for Algorithm SEF with the more accurate smoothness density estimates) to $3/2\sqrt{2} \approx 1.061$.

We mention one final improvement. The matrix to which we apply Gaussian elimination in stage 2 is “sparse”; indeed, since any integer less than n has $O(\log n)$ prime factors, the total number of non-zero entries in the matrix is $k^{1+o(1)}$. There are special algorithms for working with such sparse matrices, which allow us to perform stage 2 of the factoring algorithm in time $k^{2+o(1)}$, or

$$\exp[(2 + o(1)) \log y].$$

Setting

$$y := \exp[\tfrac{1}{2}(\log n \log\log n)^{1/2}],$$

the total running time is bounded by

$$\exp[(1 + o(1))(\log n \log\log n)^{1/2}].$$

Thus, this improvement reduces the constant in the exponent from $3/2\sqrt{2} \approx 1.061$ to 1. Moreover, the special algorithms designed to work with sparse matrices typically use much less space than ordinary Gaussian elimination (even if the input to Gaussian elimination is sparse, the intermediate matrices will not be). We shall discuss in detail later, in §18.4, one such algorithm for solving sparse systems of linear equations.

The quadratic sieve may fail to factor n, for one of two reasons: first, it may fail to find $k + 1$ relations; second, it may find these relations, but in stage 2, it finds only a trivial square root of 1. There is no rigorous theory to say why the algorithm should not fail for one of these two reasons, but experience shows that the algorithm does indeed work as expected.

15.5 Notes

Many of the algorithmic ideas in this chapter were first developed for the problem of factoring integers, and then later adapted to the discrete logarithm problem. The first (heuristic) subexponential-time algorithm for factoring integers, called the **continued fraction method** (not discussed here), was introduced by Lehmer and Powers [59], and later refined and implemented by Morrison and Brillhart [70]. The first rigorously analyzed subexponential-time algorithm for factoring integers was introduced by Dixon [35]. Algorithm SEF is a variation of Dixon’s algorithm, which works the same way as Algorithm SEF, except that it generates relations of the form (15.6) directly (and indeed, it is possible to prove a variant of Theorem 15.1, and for that matter, Theorem 15.7, for random squares modulo n). Algorithm SEF is based on an idea suggested by Rackoff (personal communication).

Theorem 15.7 was proved by Canfield, Erdős, and Pomerance [23].

The quadratic sieve was introduced by Pomerance [78]. Recall that the quadratic sieve has a heuristic running time of

$$\exp[(1 + o(1))(\log n \log\log n)^{1/2}].$$

This running time bound can also be achieved *rigorously* by a result of Lenstra and Pomerance [61], and to date, this is the best rigorous running time bound for factoring algorithms. We should stress, however, that most practitioners in this field are not so much interested in rigorous running time analyses as they are in actually factoring integers, and, for such purposes, heuristic running time estimates are quite acceptable. Indeed, the quadratic sieve is much more practical than the algorithm in [61], which is mainly of theoretical interest.

There are two other factoring algorithms not discussed here, but that should anyway at least be mentioned. The first is the **elliptic curve method**, introduced by Lenstra [60]. Unlike all of the other known subexponential-time algorithms, the running time of this algorithm is sensitive to the sizes of the factors of n; in particular, if p is the smallest prime dividing n, the algorithm will find p (heuristically) in expected time

$$\exp[(\sqrt{2} + o(1))(\log p \log\log p)^{1/2}] \cdot \operatorname{len}(n)^{O(1)}.$$

This algorithm is quite practical, and is the method of choice when it is known (or suspected) that n has some small factors. It also has the advantage that it uses only polynomial space (unlike all of the other known subexponential-time factoring algorithms).

The second is the **number field sieve**, the basic idea of which was introduced by Pollard [77], and later generalized and refined by Buhler, Lenstra, and Pomerance [21], as well as by others. The number field sieve will split n (heuristically) in expected time

$$\exp[(c + o(1))(\log n)^{1/3}(\log\log n)^{2/3}],$$

where c is a constant (currently, the smallest value of c is 1.902, a result due to Coppersmith [27]). The number field sieve is currently the asymptotically fastest known factoring algorithm (at least, heuristically), and it is also practical, having been used to set the latest factoring record — the factorization of a 200-decimal-digit integer that is the product of two primes of about the same size. See the web page `www.crypto-world.com/FactorRecords.html` for more details (as well as for announcements of new records).

As for subexponential-time algorithms for discrete logarithms, Adleman [1] adapted the ideas used for factoring to the discrete logarithm problem, although it seems that some of the basic ideas were known much earlier. Algorithm SEDL is a variation on this algorithm, and the basic technique is usually referred to as the

index calculus method. The basic idea of the number field sieve was adapted to the discrete logarithm problem by Gordon [42]; see also Adleman [2] and Schirokauer, Weber, and Denny [84].

For many more details and references for subexponential-time algorithms for factoring and discrete logarithms, see Chapter 6 of Crandall and Pomerance [30]. Also, see the web page `www.crypto-world.com/FactorWorld.html` for links to research papers and implementation reports.

For more details regarding the security of signature schemes, as discussed following Exercise 15.4, see the paper by Bellare and Rogaway [13].

Last, but not least, we should mention the fact that there are in fact *polynomial-time* algorithms for factoring and for computing discrete logarithms; however, these algorithms require special hardware, namely, a **quantum computer**. Shor [92, 93] showed that these problems could be solved in polynomial time on such a device; however, at the present time, it is unclear when and if such machines will ever be built. Much, indeed most, of modern-day cryptography will crumble if this happens, or if efficient "classical" algorithms for these problems are discovered (which is still a real possibility).