A knapsack public-key cryptosystem

Alice wants to send a secret message to Bob. Bob chooses some secret (but easy) knapsack weights $w_1, \ldots, w_k$ (a superincreasing sequence, for example). He chooses two (secret) coprime integers N, e with $N > \sum_1^k w_i$. He computes knapsack weights $h_1, \ldots, h_k$ satisfying

$$h_i = ew_i (\text{mod } N).$$

There will be infinitely many choices satisfying these conditions and Bob tries to make his weights correspond to a hard knapsack problem. Bob makes public his weights $h_1, \ldots, h_k$.

To send Bob a message consisting of k binary digits $x_1, \ldots, x_k$ Alice computes

$$M = \sum_{i=1}^{k} x_i h_i$$

and sends M to Bob.

To decrypt the message Bob finds d such that $de = 1 (\text{mod } N)$ and computes

$$\begin{aligned} dM &= \sum_{i=1}^{k} x_i (dh_i) \\ &= \sum_{i=1}^{k} x_i w_i (\text{mod } N) \\ &= \sum_{i=1}^{k} x_i w_i \quad (\text{since } N > \sum w_i). \end{aligned}$$

Bob can easily recover $x_1, \ldots, x_k$ since the weights $w_1, \ldots, w_k$ correspond to an easy knapsack problem.

The operations required to implement this system are extremely simple — much easier than in RSA or in discrete logarithm cryptosystems.

Suppose the evil Eve hacks into the system and intercepts the message M sent by Alice to Bob. She knows the weights $h_1, \ldots, h_k$ which Bob had published, but does not know e, d or N. If she wants to recover $x_1, \ldots, x_k$ she will have to solve the hard knapsack problem $\sum x_i h_i = M$.

Example Choose easy weights $w_i = 3^i$ for $1 \leq i \leq 5$, and take $N = 400 (> 3 + 9 + 27 + 81 + 243 = 363)$. Pick $e = 147$ say, and compute (easily, using the Euclidean algorithm) $d = 283$. Choose (hard ?) knapsack weights $h_i = 147 w_i (\text{mod } 400)$ as $h_1 = 41$, $h_2 = 123$, $h_3 = 369$, $h_4 = 307$, and $h_5 = 121$. These are the weights that are made public.

Alice wants to send us the message $1, 0, 0, 1, 1$, which she encodes as $M = 41 + 307 + 121 = 469$.

To decrypt this we compute $283 \times 469 = 327 (\text{mod } 400)$, and solve the easy problem $327 = 1.3^1 + 0.3^2 + 0, 3^3 + 1.3^4 + 1.3^5$ to recover the message.

2.12 Breaking the knapsack system with LLL

If we are lucky, we may be able to use LLL to break the knapsack cryptosystem. We want to find $x_i \in \{0, 1\}$ such that $\sum_1^k x_i h_i = M$.

Consider the lattice of rank $k + 1$ generated by the basis

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ h_1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ h_2 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \\ h_k \end{pmatrix}, \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ -M \end{pmatrix}.$$

A solution to $\sum_1^k x_i h_i = M$ will give a (very) short lattice vector

$$\begin{pmatrix} x_1 \\ \vdots \\ x_k \\ 0 \end{pmatrix}$$

which LLL may be able to find.

For the example of the previous section, starting with

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 41 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 123 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 369 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 307 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 121 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -469 \end{pmatrix}$$

the LLL algorithm produces the reduced basis

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -2 \\ 1 \\ 0 \\ -2 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ -2 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ -2 \\ -5 \\ -2 \\ 3 \\ 3 \end{pmatrix},$$

and the first of these does indeed recover the message M.

(Incidentally, the squared lengths of these reduced basis vectors are, in order, 3, 6, 7, 10, 8, 51. Generally we expect the lengths to be in increasing order, but it does not always happen, as this example illustrates.)

Factorisation of integers

Overview, trial division, and Fermat's method

The problem Given an integer $n > 1$ we want to find a prime factor p of n.

If we have an algorithm to do this, we can iterate it on n/p to find the complete prime factorisation of n. From now on we will assume that n is not prime, since there are good, fast, (polynomial) algorithms for primality testing — an interesting topic which we shall not go into here.

A solution: trial division
Try dividing n by 2, 3, 5, 7, ... until a factor is found. One may either test only primes as possible factors, which requires one to have a way (maybe a table) to detect primes, or one can try all odd divisors. Given that n is composite, it has a factor $p \leq \sqrt{n}$, so at most $\sqrt{n}$ divisions are necessary.

Trial division is an important benchmark against which to measure other algorithms. It is a very poor algorithm for numbers which have been specially constructed to be hard to factor, by multiplying together two large primes for example. On the other hand, for large numbers n produced randomly, trial division has an excellent chance of finding a prime divisor very quickly. Indeed for 50% of large random integers, trial division finds a factor at the very first attempt! Thus for randomly produced integers trial division up to some cut-off point may be a sensible first move.

Fermat's method
Try to solve $n = x^2 - y^2$ for integers x, y, so that $n = (x - y)(x + y)$ gives a factorization of n. Usually one searches using x, starting at $x_0 = \lceil \sqrt{n} \rceil$ and trying $x_0, x_0 + 1, x_0 + 2, \ldots$. If $n = pq$ with p small one needs to reach $x = (p + q)/2$, which will be much larger than $\sqrt{n}$. So the method will be much worse than trial division. However if p and q are close the method can be more efficient. For example, with $n = 971609 = 809 \times 1201$ one starts at $x_0 = 986$, and has to try values of x until one reaches $(p+q)/2 = 1005$; thus there are 20 trials to perform.
Fermat's method can be refined, for example by using congruences to restrict the values of x to be tried. In this case $n = 1 (\text{mod } 8)$, so that if $x^2 - y^2 = n$ the number $x^2 - 1$ will be a square (mod 8). Thus x cannot be even. Similarly $n = 2 (\text{mod } 3)$ so that $x^2 - 2$ must be a square (mod 3). This can only happen when $3 \mid x$. These consideration reduce the values of x to be tried to $x = 987, 993, 999, 1005, 1011, \ldots$, of which the fourth value is successful.

Fermat's method is now of historical interest only, but many modern approaches try to express n as a difference of squares by much more subtle methods.

The remainder of the course

For the rest of the course we will take a look at a few of the many factorisation methods that have been published. Some are true algorithms, while others have a probabilistic and/or heuristic element to them. Although our focus will be on running times, the reader should also think about the difficulties in implementation, the possibility of parallelisation, and questions of storage requirements, for example.

Most of the mathematical prerequisites have been prepared, but we will have an interlude on smooth numbers when we discuss factor-base methods.

Euler's method and its more recent variants

Euler observed that if, for a given d, one can find two different representations of n as $x^2 + dy^2$, say as

$$n = x_0^2 + dy_0^2 = x_1^2 + dy_1^2,$$

then $n \mid (y_1x_0)^2 - (y_0x_1)^2 = (y_1x_0 - y_0x_1)(y_1x_0 + y_0x_1)$. Unless we are unlucky, and one can give precise conditions under which this happens, n will divide neither of the individual factors $y_1x_0 - y_0x_1$ or $y_1x_0 + y_0x_1$, so that we will obtain a non-trivial divisor of n by computing $\gcd(n, y_1x_0 - y_0x_1)$.

For a given choice of d one would check possible values $y \leq \sqrt{n/d}$ to see whether $n - dy^2$ is a square. Although a large value of d decreases the amount of work to be done, it also decreases the chance that n will have one, let alone two, representations as $x^2 + dy^2$.

Lehmer & Lehmer's variant (1974)

Suppose that n is a product of just two prime factors. (Trial division up to $n^{1/3}$ will reduce us to this case.) Then one can guarantee that one of the following ten forms will split n via Euler's method:-

$$x^2 + y^2,\ x^2 + 2y^2,\ x^2 - 2y^2,\ x^2 + 3y^2,\ x^2 - 3y^2,$$

$$x^2 + 6y^2,\ x^2 - 6y^2,\ 3x^2 - y^2,\ 6x^2 - y^2,\ 2x^2 + 3y^2.$$

For the indefinite forms, it suffices to look at values x, y of size $O(\sqrt{n})$. In fact, for a specific n, one can reduce the number of forms to be tried to 3, by congruence considerations.

One can reduce the possible values of x, y to be tried using congruences, in the same way as with Fermat's method. At the time it was devised this method was one of the fastest in practice(!)