Public Key Cryptography

I recall thinking that this paper would be the least interesting paper that I will ever be on. *Leonard Adleman*

- The principle of a public-key cryptosystem is that it should be easy to transmit information in a secure format, but special knowledge (known as the private key) is needed to retrieve (or decode) the information.
- The information is encrypted using a public key which is available to everyone.
- The important notion is of a trapdoor or one-way function which is easy to compute but whose inverse is hard to compute. Here encryption is easy, but decryption is difficult.

RSA Cryptosystem

- Named after its inventors Ronald Rivest, Adi Shamir, and Leonard Aldeman.
- Normally we want to transmit text, made up of characters. However, the RSA system works with natural numbers, so we must first convert characters into numbers: this is easily done using the ASCII format which defines a 7-bit number for each letter of the alphabet.
- We start by choosing p and q, two distinct large prime numbers (the difficulty of cracking the code depends on the size of the numbers). Recall that it is relatively easy to decide if a number is prime. We let $n = pq$.

RSA Keys

- The private key is any number k between 1 and n which is coprime with $(p-1)(q-1)$ (for example, we could choose k to be prime); this means that $\text{hcf}(k,(p-1)(q-1)) = 1$.
- By Euclid's algorithm, there are integers a and b such that

$$ak + b(p-1)(q-1) = 1. \tag{1}$$

  We can assume that $0 < a < (p-1)(q-1)$.
- The pair of numbers (a, n) forms the public key.

RSA Encryption and Decryption

- Suppose that we have an integer M in the range 0 to $n-1$.
- To encrypt M we apply the encryption function e

$$e(M) = M^a \bmod n. \tag{2}$$

  This clearly only requires knowledge of the public key.

- We can decrypt a message using the private key k:

$$M = (e(M))^k \bmod n. \tag{3}$$

RSA Example

- Take $p = 53$ and $q = 61$, so that $n = 3233$ and we require k coprime to $(p-1)(q-1) = 3120 = 2^4 \times 3 \times 5 \times 13$. Let us choose $k = 1013$ (which is actually prime).
- Using Euclid's algorithm we find that

$$77k - 25 \times 3120 = 1$$

  so $a = 77$ and the public key is $(77, 3233)$.
- A number M between 0 and 3233 is encrypted as M^{77} mod 3233. For example, if $M = 10$ this is 10^{77} mod $3233 = 2560$.
- We decrypt this by computing 2560^{1013} mod $3233 = 10$.
- If we wanted to transmit a string of decimal digits, we would have to split it into blocks of length 3 to ensure that every number was less than 3233. A string of binary digits could be split into blocks of length 11, since $2^{11} = 2408 < 3233$.

Efficient Computation of Modular Powers

- It is very time consuming to compute powers directly. For example, calculating M^{77} by working out M^2, M^3, M^4, ... requires 76 multiplications modulo 3233.
- Instead, we express a as a binary number (here $77 = 1001101_2$) and use multiplications by M and squaring instead. For each binary digit, we square the number and multiply by M if there is a 1.

In detail, to compute $L = M^k$:

1. Write k as a binary number with d bits; the most significant bit is 1. We number the bits from most to least significant.

$$a = b_1 \dots b_d \tag{4}$$

2. Compute $L = M^2$. Set the index $i = 2$.
3. If $b_i = 1$, let $L := L \times M$.
4. If $i < d$, let $L := L^2$, $i := i + 1$ and go to step 3.

Suppose that we want to compute M^{10}. The binary representation of 10 is 1010, which requires 4 bits. So we calculate

$$\begin{array}{cccccc} & b_1 & & b_2 & & b_3 & & b_4 \\ M & \to & M^2 & \to & M^4 & \to & M^5 & \to & M^{10} & \to & M^{10} \end{array}$$

RSA Security

- We have said that this encryption method is one-way. What we mean by this is that given the public key (a, n), it is prohibitively difficult to compute the private key k.
- This is because it is believed to be essentially equivalent to finding the two prime factors p and q. Certainly, knowing p and q, it is easy to work out k, since

$$ak = 1 \bmod (p-1)(q-1).$$

- At present there are algorithms which will factorise numbers of a little more than 100 digits on powerful computers, so it seems safe to use prime numbers p and q of about this size, giving n of about 200 digits.

RSA Security II

There are two concerns:

1. As computers become more powerful, the size of practically solvable factorisation problems will increase. This is not too bad, as we can simply slightly increase the length of primes to compensate.
2. Someone will discover a fast method for factorising n. This could completely invalidate the security of the RSA algorithm, since if factorising is nearly as fast as primality testing, it would be nearly as fast to decode a message as to set up the code in the first place.

Summary

1. Modular arithmetic has applications in random number generators, cryptography, and error-correcting codes.
2. Modular addition, subtraction and multiplication use clock arithmetic.
3. If the modulus is not prime, then the product of two non-zero values can be zero.
4. Fermat's little theorem, $a^{p-1} = 1 \bmod p$, gives us a way of finding the multiplicative inverse of a modular number.
5. Ada contains modular types.
6. A pseudo-random number generators is a deterministic algorithm that generates a sequence of numbers that appear to be random under statistical tests.
7. The algorithm $I_{j+1} \equiv aI_j \bmod m$ can be used as a RNG for suitable choice of a and m.
8. A public-key cryptosystem allows anyone to encode information (with the public key), but only privileged people can decode information (with the private key).
9. The security of the RSA algorithm is based on the difficulty of finding the prime factorisation of large numbers.

Floating Point Arithmetic

The problems of representing real numbers, rounding and the errors it introduces.

The usual way to represent real numbers in a computer is by means of a binary ‘floating point’ notation. This is similar to ‘scientific notation’ with some technical alterations to represent negative numbers and zero.

Scientific Notation

- In radix 10, the scientific notation for a positive real number a takes the form $a = b \times 10^c$ where $1 \leq b < 10$ and c is an integer.

$$3279.5524 = 3.279\ 552\ 4 \times 10^3$$

$$0.000\ 000\ 000\ 000\ 197\ 704 = 1.977\ 04 \times 10^{-13}$$

- To find the scientific notation for a number, the decimal point is moved to the left or right until the result lies between 1 and 10, which gives the value of b.
- The number of places the decimal point was moved to the left is recorded as c (and if it had to be moved to the right, c is the corresponding negative integer, or if it didn't move at all, then $c = 0$).

Exercise

Express the following numbers in scientific notation:

1. $9\,800\,270 =$
2. $0.001\,010\,1 =$
3. $2.775\,13 =$

Solution

Express the following numbers in scientific notation:

1. $9\,800\,270 = 9.80027 \times 10^{6}$
2. $0.001\,010\,1 = 1.0101 \times 10^{-3}$
3. $2.775\,13 = 2.77513 \times 10^{0}$

IEEE Floating Point Standard (32 bit)

- Given a positive number a, write both its integer and fractional parts in binary form, and transform this by moving the binary decimal point to the left or right until $a = b \times 2^c$, where $1 \leq b < 2$ and c is an integer.

  1. The first bit is 0.
  2. The next eight bits store the binary form of $c + 127$ (provided $-126 \leq c \leq 128$). These are called the exponent bits.
  3. The remaining 23 bits store the first 23 places of the fractional part of b. These are called the mantissa bits.

- For negative reals, we use 1 as the first bit instead of 0, while 0 is represented by the all zero string. The significant bits are those in the mantissa.

Machine Numbers

Given a fixed number of bits for the exponent and mantissa, there is a set $A \subseteq \mathbb{R}$ of exactly representable numbers; the elements of A are known as machine numbers. Not all real numbers are machine numbers.

- There are some whose magnitude is too big, i.e. the exponent can't be represented at the positive end; in the example, this occurs if $c > 128$. This is called exponent overflow.
- There are some whose magnitude is too small, i.e. the exponent can't be represented at the negative end; in the example, this occurs if $c < -127$. This is called exponent underflow.
- There are some whose fractional part cannot be represented in the mantissa; in the example, this occurs if the fractional part is not an integer multiple of 2^{-23}. This requires rounding.

Rounding

We define the rounding function $\mathrm{rd} : \mathbb{R} \to \mathbb{R}$ as follows, where t bits are used in the mantissa. If

$$a = b \times 2^c,$$

with $1 \leq |b| < 2$,

$$|b| = 1.\beta_1 \ldots \beta_t \beta_{t+1} \ldots \qquad \text{where } \beta_1 = 1,$$

then let

$$b' = \begin{cases} 1.\beta_1 \ldots \beta_t & \text{if } \beta_{t+1} = 0 \\ 1.\beta_1 \ldots \beta_t + 2^{-t} & \text{if } \beta_{t+1} = 1. \end{cases}$$

Then we define rd by

$$\mathrm{rd}(a) := \mathrm{sign}(a) \cdot b' \times 2^c.$$

This is just a formalisation of the usual rounding up/down rule.

Machine Precision

- The machine precision is given by eps $:= 2^{-t}$, so for 32-bit numbers, machine precision is 2^{-23}. Then

$$\text{rd}(a) = a(1+\epsilon), \tag{5}$$

  where $|\epsilon| \leq \text{eps} = 2^{-t}$. This expresses the fact that rounding introduces an error of relative size ϵ.

- Unfortunately the rounded value rd(x) may not be in A (that is, may not be a machine number) because the exponent may not fit.

- The IEEE standard mandates that if there is exponent overflow the value should be rounded to a special number Inf $\notin A$.

- If there is exponent underflow, the exponent should be frozen at its smallest value, but more zeroes allowed in the mantissa. As the mantissa becomes smaller it should be allowed to tend to zero.

Floating Point Operators

- Because floating point numbers are not the same as the real numbers (due to the limited precision), we must define new arithmetic operators: $x +^* y := \mathrm{rd}(x + y)$ and similarly for subtraction, multiplication and division.
- These elementary operations are known as flops (short for 'floating point operations').
- However, these new operations do not satisfy the usual laws of arithmetic. For example, $x +^* y = x$ if $|y| < \mathrm{eps}|x|$, for $x, y \in A$.
- Another way to view eps is that it is the smallest positive machine number that, when added to 1, gives an answer that differs from one.

$$\mathrm{eps} = \min\{g \in A \mid 1 +^* g > 1 \quad \text{and} \quad g > 0\}.$$

- In fact, pretty well any flop should be thought of as introducing $\mathrm{eps} = \epsilon_m$ in relative error.

Effect of Rounding Error

- An error of size eps may not look bery big.
- For 32-bit floating point numbers this is $2^{-23} \approx 1.19 \times 10^{-7}$, while for double precision numbers this is $2^{-52} \approx 2.22 \times 10^{-16}$.
- However, during a long computation rounding errors can accumulate until the answer is so inaccurate as to be useless.

Now read the example.

Comment on Example

- Ironically, the fact that some programmers had noticed this problem and had 'improved' their part of the code, while others had not, meant that the inaccuracies did not cancel and made the problem significantly worse.
- Using the time since booting the system makes the problem much worse, since the rounding error on 1/10 is multiplied by a large value. There seems to be no good reason for this choice.

Effect of Rounding Error

- We shall write u for the true real number, and $\tilde{u}$ for the approximation introduced by rounding (and similarly for v).
- The absolute error in u is

$$\Delta u := \tilde{u} - u \tag{6}$$

- The relative error in u is

$$\epsilon_u := \frac{\Delta u}{u}, \tag{7}$$

  if $u \neq 0$.
- We are interested in the relationship between the relative errors of the inputs (u and v) and the relative error of the output.

Rounding Error Analysis of Multiplication

- $y = f(u, v) := u \times v$. Then

$$\epsilon_y \doteq \epsilon_u + \epsilon_v.$$

- This means that the relative error in the result is the sum of the relative errors on the inputs. Thus if we do a lot of multiplications, the error increases slowly.
- The technical term for this is that multiplication is well-conditioned.

Rounding Error Analysis of Addition

- $y = f(u, v) := u + v$. Then

$$\epsilon_y \doteq \frac{u}{u+v}\epsilon_u + \frac{v}{u+v}\epsilon_v,$$

  if $u + v \neq 0$.

- This is OK if u and v have the same sign. In this case, $|\epsilon_{u+v}| \leq \max\{|\epsilon_u|, |\epsilon_v|\}$ and the problem is well-conditioned (the errors are damped).

- If u and v have opposite signs, at least one of

$$\left|\frac{u}{u+v}\right| \qquad \text{and} \qquad \left|\frac{v}{u+v}\right| > 1,$$

  and so errors are magnified. The closer to 0 the value of $u + v$ lies, the worse conditioned the computation.

Subtractive Cancellation

- We can create catastrophically bad errors with addition; this effect is called subtractive cancellation.
- Consider a machine with floating point arithmetic accurate to 5 decimal places. Let $u = 0.37214\,48693$ and $v = 0.37202\,14371$. Then the machine numbers are $\tilde{u} = 0.37214$ and $\tilde{v} = 0.37202$.
- It follows that

$$\tilde{u} - \tilde{v} = 0.00012 \qquad u - v = 0.00012\,34322.$$

We can see that the computed result only contains 2 significant figures, not the full five. This is confirmed by the size of the relative error

$$\frac{|\tilde{u} - \tilde{v}|}{|u - v|} \approx 3 \times 10^{-2}.$$

Summary

1. The IEEE standard floating point notation for a real number a is $a = b \times 2^c$, where $1 \leq b < 2$ and c is an integer. b is the mantissa and c is the exponent.
2. Given a fixed number of bits for the exponent and mantissa, there is a set a of exactly representable machine numbers; the elements of A are known as machine numbers.
3. Rounding is used to convert general real numbers to machine numbers.
4. Machine precision is given by 2^{-t}, where t bits are used in the mantissa. For 32-bit numbers, machine precision is 2^{-23}.
5. Floating point operations (flops) are defined by rounding mathematical operations, e.g. $x +^* y := \mathrm{rd}(x + y)$.
6. Multiplication is well-conditioned: relative errors of the arguments are summed.
7. Addition is ill-conditioned: subtractive cancellation can cause catastrophic errors.