2.3 Public Key Cryptography

This project requires an understanding of the Part IA course Numbers and Sets.

1 The RSA Algorithm

The essence of Public Key Cryptography is to make it easy for people to send securely encrypted messages to one another without the need for each pair to agree beforehand on a secret code. Encryption and decryption is done via a publicly known algorithm which makes use of a *key*. Each person privately constructs two keys K_1 and K_2 which are, in effect, inverses of each other; that is to say, a message encrypted with K_1 can be decrypted using K_2. The person then publishes K_1 to the whole community — it becomes that person's *public key*. The key K_2 is kept (very) private. Then, to send a message to any given person, the message is encrypted using their public key. Only that person can decrypt the message because only that person has the decryption key K_2.

In order for a public key algorithm to be effective, it must be feasible for a person to construct two inverse keys K_1 and K_2, but it must be infeasible for anyone to work out the value of K_2 given only the value of K_1.

In the RSA system (so called after its inventors, Rivest, Shamir and Adleman) an individual follows this recipe. Choose two large primes p and q and compute $n = pq$. Then choose a number $e < n$ which is coprime to $\varphi(n) = (p-1)(q-1)$. Then find a number d such that $ed \equiv 1 \pmod{\varphi(n)}$. The public key is the pair (n, e) and the private key is (n, d). A message m sent to this person would be encrypted as $c = m^e \pmod{n}$. The individual for whom the message was intended would decrypt c as $m = c^d \pmod{n}$.

The RSA algorithm works because it is feasible for a user to find a couple of 100-digit numbers which are almost certainly prime, but it is thought to be impossible to factorize a 200-digit number in a reasonable time. Factorization of n would enable an enemy to find d from e in just the same way that the user did. There are other possible attacks but they appear to be (though are not proved to be) as difficult as factorization.

2 Working with integers

The quantities of interest in this project are natural numbers. MATLAB, unfortunately, is designed for those who think that real numbers are the only really natural numbers. This has an effect on how easy it is to get MATLAB to do integer arithmetic and on how large the numbers can be before errors start to occur.

Computers typically store a number as a string of 32 or 64 binary digits (bits), which can, if desired, represent integers up to 2^{32} or 2^{64}. If the computer understands that we are working only with integers, then the representation is very efficient, and moreover, if it is asked to compute, say, 14/3 it will respond with 4, this being the correct integer value. In other words, the machine will perform modular arithmetic simply and with complete accuracy, which is ideal.

MATLAB, though, with its incorrigible real-world view, uses the 64 bits to store what it believes is a real number. This means both that the space available for the integer part is limited, and that rounding errors will occur. If you ask for the value of 14/3 it will produce a reasonable but incorrect result. You can force MATLAB to recognize integers via the `int32` or `int64` commands (see the CATAM manual). However the results are not always what you might expect, and for

this project it is suggested that you go with the flow and just allow MATLAB to think it is working with reals.

The only practical consequence of the above discussion that needs to be borne in mind is that MATLAB can handle integers in this way only up to about 15 decimal digits. (Try `eps(10^15)` and `eps(10^16)`. Try also `10^8 * 10^8 - (10^16-1)`.) So you must avoid calculations involving integers exceeding 10^{15}. This is a pity, because it is easy to crack RSA with numbers of this size. The adventurous might like to find out how to handle larger numbers in MATLAB; trying RSA on 50-digit numbers brings home much more vividly what is going on. But this remark is for just for interest, and you will get no extra credit for pursuing it.

You will find it helpful to type `format longg` at the beginning of a MATLAB session: this encourages MATLAB to print integers larger than 10^9 properly, rather than in engineering format.

3 Prime Numbers and Factorization

The easiest way to test a small number n for primality is by *trial division*: divide by every number up to n. The MATLAB function `mod` is useful for this: `mod(a,b)` gives the value of a modulo b. The builtin functions `isprime()`, `factor()` and `primes()` should not be used in the next two questions except, if desired, for checking answers.

> **Question 1** This method takes a while for a 10-digit number; what simple modification will speed it up? Write a function to apply this method (and exhibit sample output).

In subsequent questions, an "arithmetic operation" can be taken to be the addition, subtraction, multiplication or division of two integers.

> **Question 2** Modify your improved algorithm from Question 1 so as to find the complete prime factorization of a number n. Try your algorithm on a few examples.
>
> Your algorithm may be much quicker for some types of numbers than for others of similar size. Estimate the complexity of your algorithm — that is, estimate the number of arithmetic operations needed in the worst case, as a function of n. (You may wish to consider first some extreme cases.) Can you prove that your estimate is correct?

4 Linear Congruences

One way to find the highest common factor of two numbers is to factorize them both and then form the product of the common prime factors.

Euclid's Algorithm is a more efficient way to compute the highest common factor of two numbers. Importantly, it also provides a means to find integers u and v such that $au + bv = \mathrm{hcf}(a, b)$.

> **Question 3** Implement Euclid's Algorithm. Thereby find the highest common factor of each of the following pairs of numbers, and express it as a linear combination of the original two numbers.
>
> 1 996 245 783 and 192 784 863 2 825 746 811 and 758 295 345
> 249 508 543 104 and 338 063 357 376 249 508 543 140 and 338 063 357 367

Question 4 Describe clearly how Euclid's algorithm can be used to find all of the solutions in the unknown x to the linear congruence $ax \equiv b \pmod{m}$.

Question 5 Implement a routine to solve the linear congruence $ax \equiv b \pmod{m}$.

Find all solutions to each of the following congruences. If none exist, state why not.

$$\begin{aligned} 146295x &\equiv 2017 \pmod{313567} \\ 93174x &\equiv 2015 \pmod{267975} \\ 113314x &\equiv 2014 \pmod{660115} \end{aligned}$$

5 Finding Inverses and Breaking RSA

Let $n = pq$ where p and q are primes. Then $\varphi(n) = (p-1)(q-1)$ where $\varphi(n)$ is Euler's function. If e is coprime to $\varphi(n)$ then there exists a d such that $ed \equiv 1 \pmod{\varphi(n)}$. The program you developed for Question 5 should enable you to find such a d for a given e.

Question 6 Given n and e, but not p or q, approximately how many arithmetic operations does your program need to find p and q? (Remember to justify your answers.)

Given p and q, and hence $\varphi(n)$, approximately how many operations are needed to find d?

Question 7 Write a program to compute the private decryption key from a given public encryption key. Find the decryption keys corresponding to the following:

$$\begin{array}{ccc} (1\,764\,053\,131, 103\,471) & (1\,805\,760\,301, 39\,871\,477) & (9\,976\,901\,028\,181, 837\,856\,358\,917) \\ (1\,723\,466\,867, 692\,581\,937) & (6\,734\,071\,952\,813, 2017) & (1\,603\,982\,333, 927\,145) \end{array}$$

6 Decrypting RSA messages

Question 8 Write a program to convert an encrypted number $c = m^e \pmod{n}$ into the original $m = c^d \pmod{n}$, where $0 \leqslant m < n$ is some integer.

State, with justification, the greatest number of digits that n can have for which your program can be trusted to work.

My public key is $(937513, 638471)$. Our community has agreed to use the encoding $00 \leftrightarrow$ `space`, $01 \leftrightarrow$ `a`, $\ldots$, $26 \leftrightarrow$ `z`, $27 \leftrightarrow$., $28 \leftrightarrow$:, $29 \leftrightarrow$ ′, and to encrypt blocks of 3 letters at a time (so that, for example, the message "`have a nice day`" is encoded as the five numbers 080122 050001 001409 030500 040125 before encryption).

Question 9 I receive the encrypted message

179232	006825	263565	126615	474921	750809	900050	009287
554344	413204	757176	066356	716784	382286	696566	610518
510930	459403	922484	390971	773831	655925	633419	519880

What is the message?

[You do not need to write programs to convert a string of characters into the appropriate sequence of integers or vice versa.]

