

Lab 2.6 Fundamentals (Primes, GCD, Random Numbers and Exponentiation)

Many of the key concepts in cryptography are based on number theory which is the study of integers, with a special focus on divisibility. The main classifications for numbers are integers, rational numbers, real numbers and complex numbers. In maths we define these as:

- Integers can be positive or negative numbers and have no fractional part. They are represented with the \mathbb{Z} symbol $\{\dots-2, -1, 0, +1, +2, \dots\}$.
- Rational numbers are fractions (\mathbb{Q}).
- Real numbers (\mathbb{R}) include both integers and rational numbers, and any other number that can be used in a comparison.
- Prime numbers (\mathbb{P}) represent the integers which can only be divisible by itself and unity.
- Natural numbers (\mathbb{N}) represent positive numbers which are integers $\{1, 2, \dots\}$

A Prime number test

A prime number is a value which only has factors of 1 and itself. Prime numbers are used fairly extensively in cryptography, as computers struggle to factorize them when they are multiplied together. The simplest test for a prime number is to divide the value from all the integers from 2 to the value divided by 2. If any of the results leaves no remainder, the value is a prime, otherwise it is composite. We can obviously improve on this by getting rid of even numbers which are greater than 2, and also that the highest value to be tested is the square root of the value.

So if $n = 37$, then our maximum value will be \sqrt{n} , which, when rounded down is 6. So we can try: 2, 3, and 5, which of none of these divide exactly into 37, so it is a prime number. Now let's try 55, we will then be 2, 3, 5 and 7. In this case 5 does divide exactly in 55, so the value is not prime.

Another improvement we can make is that prime numbers (apart from 2 and 3) fit into the equation of:

$$6k \pm 1$$

where $k=0$ gives 0 and 1, $k=1$ gives 5 and 7, $k=2$ gives 11 and 13, $k=3$ gives 17 and 19, and so on. Thus we can test if we can divide by 2 and then by 3, and then check all the numbers of $6k \pm 1$ up to \sqrt{n} .

 **Web link (Prime Numbers):** <http://asecuritysite.com/encryption/isprime>

No	Description	Result
1	Using the equation of $6k \pm 1$. Determine the prime numbers up to 100:	Prime numbers:

2	Implement a Python program which will calculate the prime numbers up to 1000:	Define the highest prime number generated:
----------	---	--

A prime sieve creates all the prime numbers up to a given limit. It progressively removes composite numbers until it only has prime numbers left, and it is the most efficient way to generate a range of prime numbers. The following provides a fast method to determine the prime numbers up to a give value (test):

```
import sys
test=1000
if (len(sys.argv)>1):
    test=int(sys.argv[1])
def sieve_for_primes_to(n):
    size = n//2
    sieve = [1]*size
    limit = int(n**0.5)
    for i in range(1,limit):
        if sieve[i]:
            val = 2*i+1
            tmp = ((size-1) - i)//val
            sieve[i+val::val] = [0]*tmp
    return [2] + [i*2+1 for i, v in enumerate(sieve) if v and i>0]
print sieve_for_primes_to(test)
```

No	Description	Result
1	Implement the Python code given above and determine the highest prime number possible in the following ranges:	Up to 100: Up to 1,000: Up to 5,000: Up to 10,000:

The Miller-Rabin Test for Primes is an efficient method in testing for a prime number. Access the following page, and download the Python script.

<http://asecuritysite.com/encryption/rabin>

Using this determine the following:

No	Description	Result
1	Which of the following numbers are prime numbers:	Is 5 prime? Yes/No Is 7919 prime? Yes/No Is 858,599,509 prime? Yes/No Is 982,451,653 prime? Yes/No Is 982,451,652 prime? Yes/No

B GCD

GCD is known as the greatest common divisor, or greatest common factor (gcf), and is the largest positive integer that divides into two numbers without a remainder. For example, the GCD of 9 and 15 is 3. It is used in many encryption algorithms, and a sample algorithm to determine the GCD of two values (a and b) is given on:

<http://asecuritysite.com/encryption/gcd>

No	Description	Result
1	Write a Python program to determine the GCD for the following:	4105 and 10: 4539 and 6:
2	Two numbers are co-prime if they do not share co-factors, apart from 1, which is $\text{gcd}(a,b)=1$. Determine if the following values are co-prime:	5435 and 634: Yes/No 5432 and 634: Yes/No

C Modulus and Exponentiation

The **mod** operator results in the remainder of an integer divide. For example 31 divided by 8 is 3 remainder 7, thus $31 \bmod 8$ equals 7. Often in cryptography the mod operation uses a prime number, such as:

Result = $\text{value}^x \bmod (\text{prime number})$

For example, if we have a prime number of 269, and a value of 8 with an x value of 5, the result of this operation will be:

Result = $8^5 \bmod 269 = 219$

With prime numbers, if we know the result, it is difficult to find the value of x even though we have the other values, as many values of x can produce the same result. It is this feature which makes it difficult to determine a secret value (in this case the secret is x).

Exponentiation ciphers use a form of:

$$C = M^e \bmod p$$

to encrypt and decrypt a message (M) using a key of e and a prime number p .

No	Description	Result
1	What is the result of the following:	$8^{13} \bmod 271$: $12^{23} \bmod 973$:

2	<p>Implement a Python program which will determine the result of:</p> <p>$M^e \bmod p$</p> <p>The program should check that p is a prime number.</p>	<p>Is the result of $8^5 \bmod 269$ equal to 219?</p> <p>Yes/No</p>
3	<p>Now provide the following:</p> <p>(a) message = 5, e=5, p = 53. Ans: 51</p> <p>(b) message = 4, e=11, p = 79. Ans: 36</p> <p>(c) message = 101, e=7, p = 293. Ans: 176</p> <p>An outline of the Python code is:</p> <pre>message = raw_input('Enter message: ') e = raw_input('Enter exponent: ') p = raw_input('Enter prime ') cipher = (int(message) ** int(e)) % int(p) print cipher</pre>	<p>Have you proven the answers</p> <p>(a) Yes/No</p> <p>(b) Yes/No</p> <p>(c) Yes/No</p>

D Random numbers

Within cryptography random numbers are used to generate things like encryption keys. If the generation of these keys could be predicted in some way, it may be possible to guess it. The two main types of random number generators are:

- **Pseudo-Random Number Generators (PRNGs).** Repeats after a given time. Fast. They are also deterministic and periodic, so that the random number generation will eventually repeat.
- **True Random Number Generators (TRNGs).** This method is a true random number such as for keystroke analysis. It is generally slow, but is non-deterministic and aperiodic.

Normally simulation and modelling use PRNG, so that the values generated can be repeated each time, while cryptography, lotteries, gambling and games use TRNG, as each value which is selected at random should not repeat or be predictable. In the generation of encryption keys for public key encryption, a user is typically asked to generate some random activity with their mouse pointer. The random number is then generated on this activity.

Computer programs often struggle to generate TRNG, and hardware generators are sometimes used. One method is to generate a random number based on low-level, statistically random "noise" signals. This includes things like thermal noise, and a photoelectric effect.

 **Web link (Random number):** <http://asecuritysite.com/encryption/random>

Linear Congruential Random Numbers

One method of creating a simple random number generator is to use a sequence generator of the form:

$$X_{i+1} \leftarrow (a \times X_i + c) \bmod m$$

Where a , c and m are integers, and where X_0 is the seed value of the series.

If we take the values of $a=21$, $X_0=35$, $c=31$ and $m=100$ we get a series of:

66 17 88 79 90 21 72 43 34 45 76 27 98 89 0 31 82 53
--

Using this example we get:

$(21 \times 35 + 31) \bmod 100$ gives 66

$(21 \times 66 + 31) \bmod 100$ gives 17

$(21 \times 17 + 31) \bmod 100$ gives 88

and so on.

 **Web link (Linear congruential):** <http://asecuritysite.com/encryption/linear>

No	Description	Result
1	Implement the Python code given above. Using: $a=21$, seed=35, $c=31$, and $m=100$, prove that the sequence gives 66 17 88 79 90	Does it generate this sequence: Yes/No
2	Determine the sequence for: $a=22$, seed=35, $c=31$, and $m=100$.	First four numbers of sequence:
3	Determine the sequence for: $a=954,365,343$, seed=436,241, $c=55,119,927$, and $m=1,000,000$.	First four numbers of sequence:
4	Determine the sequence for: $a=2,175,143$, seed=3553, $c=10,653$, and $m=1,000,000$.	First four numbers of sequence: