'Information Security' and 'Information Security & Network Security' – Lab Session 1

David Weston

1 Getting Started

The aim of this laboratory session is to provide hands on experience of some of the essential concepts behind performing cryptography on a computer.

All you need for this lab session is a web browser. This lab session consists of 5 tasks.

The webpages used in this lab session have been adapted from:

PK Yuen, *Practical Cryptology and Web Security*, Prentice Hall 2005. ISBN-10: 0321263332

2 Numerical Representation of Messages

Sophisticated approaches to cryptography concern themselves with transforming an integer into another integer. How does this help us encrypt a message? The answer lies in the fact that we can convert text into a sequence of numbers or indeed one very large number.

One easy way to accomplish this is to realise that the computer actually already stores text in a numeric format. The industry standard way to convert text into numbers is by using Unicode. Before we describe Unicode, we briefly review binary numbers.

2.1 Binary Numbers

Strictly speaking your computer operates by manipulating binary numbers. Binary numbers contain only two symbols 0,1. In contrast to the decimal numbers we use in everyday life where we use ten symbols. Here are some examples:
0 (decimal) = 0 (binary)
1 (decimal) = 1 (binary)
2 (decimal) = 10 (binary)
3 (decimal) = 11 (binary)
255 (decimal) = 11111111 (binary)
We say that 255 (decimal) has 3 digits, likewise we say that 11111111 (binary) has 8 bits.

2.2 Unicode

Unicode uses 16 bits to encode multiple alphabets from around the world. It also includes encodings for punctuation, some geometric shapes and technical symbols. Furthermore special functions are also included, which when you try to print them, will perform a function and not print a character. Some examples are:
The letter 'A' has the Unicode value 65(decimal), 0000000001000001 (binary)
The letter 'a' has the Unicode value 97(decimal), 0000000001100001 (binary)
The registered symbol '®' has the Unicode value 174 (decimal), 0000000010101110 (binary)
The backspace character (this will move the cursor) has the Unicode value 8(decimal), 0000000000001000 (binary).

2.3 Task 1

Go to the following webpage (click the link):
`http://www.dcs.bbk.ac.uk/~dweston/infosec/Lab1/NumericalRep.htm`
This web page will accept a character and will display its corresponding Unicode value in both decimal and binary.
Try out some lower/upper case characters and familiarise yourself with the idea of binary and decimal numbers representing individual characters.

This web page will also accept a short message and convert all the characters contained in the message into number representations. Try inputting some words.

The bottom 2 boxes attempt to construct just one number to represent the entire message. It does this by appending the bit representations together. The decimal representation of this very large binary number is shown in the last box. Why do you think we can append the binary representation of the letters together, but not the decimal?

Note: for messages greater than 4 characters long this decimal value is so large that the computer will display it in scientific notation. Can you think why? For our purposes we need to know the number exactly, so keep the message to at most 4 characters!

3 Transforming Numbers

We have seen how we can convert text into numbers. The next step on our road towards encryption involves transforming those numbers.

3.1 Bitwise Operations

The first type of transformation we shall look at involve, so called, bitwise operations. These are operators that act directly on binary representations. We have already seen definitions for AND, OR, XOR and NOT in the lectures.

3.2 Task 2

Go to the following webpage:
`http://www.dcs.bbk.ac.uk/~dweston/infosec/Lab1/BitwiseOps.htm`
Familiarise yourself with the bitwise operators by entering your choice of binary numbers and examining the output.

3.3 Modular Arithmetic

The modulus operator is, in fact, something that we are all familiar with. In the case where we consider positive integers, the result is simply the remainder after a division.
13 mod 3 = 1 (3 divides into 13 four times leaving a remainder of 1)
2 mod 3 = 2 (3 divides into 2 zero times leaving a remainder of 2)
The modulo operator has some surprising properties that are useful for cryptography (and not just cryptography, it is used pseudo-random number generation for example).

3.4 Task 3

Review the lecture notes from last week regarding modulus.
Go to the following webpage:
`http://www.dcs.bbk.ac.uk/~dweston/infosec/Lab1/Modular.htm`

Satisfy yourself that, for three integers a,b and n

$$a \times b \bmod n = (a \bmod n) \times (b \bmod n) \bmod n$$

For example, let a=5, b=4 and n=3.
The left side of the equation becomes:
$5 \times 4 \bmod 3 = 20 \bmod 3 = 2$
The right side of the equation becomes:
$(5 \bmod 3) \times (4 \bmod 3) \bmod 3 = 2 \times 1 \bmod 3 = 2 \bmod 3 = 2$

Note: in the exam next year, you will be expected to be able perform modulus calculations, so you might wish to practice on the calculator you intend to take to the exam.

4 Ceasar Codes

In the lectures we have already discussed the Ceaser cipher. We shall now take a look at an actual implementation that uses Unicode.

4.1 Task 4

First familiarise yourself with the Ceaser cipher described in the lecture notes, then go to the following webpage:
`http://www.dcs.bbk.ac.uk/~dweston/infosec/Lab1/Caesar.htm`
This version is slightly different to the one in the lecture notes in that in only accepts only lower case characters. In addition it allows for a user specified shift rather than just a shift of 3.

Type in a message of your choice and shift by 3
Copy the resulting encrypted text and paste it into the message textbox, then shift by -3. Do you get the original message?
Try out other messages and shift values.

4.2 Task 5

We can build a Ceasar cipher based on the whole of the Unicode set. This means we can encrypt/decrypt a message containing any possible character.
`http://www.dcs.bbk.ac.uk/~dweston/infosec/Lab1/CaesarUnicode.htm`

Type in a message of your choice and shift by 3
Copy the resulting encrypted text and paste it into the message textbox, then shift by -3. Do you get the original message?

Type in the message 'xyz' and shift by 3. Notice that shifting 'z' by 3 does not wrap around to the beginning of the alphabet. This is unlike the version of the Ceaser cipher from our lecture notes.

The algorithm on the webpage is doing the following:
For each character in the message:
1) Convert to Unicode number
2) Add the shift value to this number
3) Return the Unicode symbol corresponding to the updated number

Although very simple, this algorithm has the potential to fail to display correctly.
Type in the message 'xyz' and shift by 39

Copy the resulting encrypted text and paste it into the message textbox, then shift by -39. Do you get the original message?

Can you explain why this happens?