Suggestions for a unit on Cryptography

The following outlines a set of tasks that could form the basis for a unit of work set within the context of cryptography.

How the tasks are constructed would depend on the age group at which the unit is aimed and the learning objectives of each task. For example, consideration should be given to how a task will be presented in relation to the role it plays in cryptography, the degree of scaffolding, and what, if any, theory should be included.

How the tasks are sequenced will depend on how concepts and skills are to be developed, in particular those relating to algorithm development and data representation. For example, some of the tasks may be better suited to the concepts and techniques relating to arrays while others may be more appropriate for basic programing language acquisition.

Finally and I think most importantly for pre-VCE classes, consideration needs to be given to the ‘hooks’ that can be used to engage students, not just in the overall topic, but also in each individual task. For example, if students were writing a decryption module I think it would be useful if students could access an encryption program with which they could send encrypted messages to each other over the LAN. I know that could open up a can of worms, but I think it given them a reason to try to get their decryption module working. Alternatively, an encrypted file could be set up somewhere for students to try to decrypt with some sort of challenge attached. Suitable video material would also be useful.

**Ideas for Tasks**

* The most basic of encryption systems is the use of substitution ciphers. Three methods could be explored: a letter for letter substitution alphabet (a=f, b=y, …); a code word is used to create the substitution alphabet (xylenolabcdf…); the substitution alphabet is obtained by rotating the alphabet **n** places.

Programming could be for encryption and decryption modules. Decryption could be first with lots of scaffolding, then encryption but expecting more independence by using decryption as a model.

Concepts and skills: keys and the issues surrounding how to get the key to authorised receivers; arrays as data structures; writing to and reading from an array; iterating through an array in one dimension; searching an array; file writing/reading; iteration through strings; removal of characters from strings; functions. Use of the ‘rotated alphabet’ could also allow manipulation at the bit level by, for example, adding **n** to the ASCII code.

* Transposition ciphers could also be investigated, where the order of the plain text characters is rearranged to produce the cipher text.

Programming could follow a plan similar to above or, if this was a follow up from that, then more independence could be expected.

Concepts and skills: similar to above, but 2-D arrays could be more extensively explored, including iteration through two dimensions; ideas about programming code re-use could also be touched on.

* Character frequency analysis, counting how many times each English letter appears in a significant amount of text, and possibly extended to include short words.

Once a module is written to determine character frequency of a plain text file, it could be adapted to read cipher text and display statistics which assist in code-breaking. Student could then try to ‘crack’ encrypted text, or try to determine each other’s key as used in one of the above.

Concepts and skills: selection structure; data types (character, integer, floating point); character comparison versus numeric comparison; iteration through text; 1-D arrays and/or associative arrays; data display; code-breaking; arithmetic operations (to determine % frequencies).

(This could also lead to text analysis such as: was this written by Shakespeare?)

* Prime number proving perhaps leading to prime number finding, for use in public-key cryptography.

Students could be given the challenge of writing a prime number proving program themselves, based on their knowledge of prime numbers (Year 7 Maths, if not earlier). Alternatively, a program to generate the first n-primes could be set as a task. Both approaches could lead to an Alistair-type analysis when very large primes are required.

Concepts and skills: creating and implementing an algorithm; poor, good and better algorithms; principles of public-key cryptography; control structures.

* Greatest Common Divisor/Highest Common Factor finding, for use in public-key cryptography.

There are two simple ‘by-hand’ methods: finding the set of common prime factors and Euclid’s algorithm. Students will possibly need to be shown a worked example of each, but could then be left to devise a working algorithm. The first technique could build on the prime number work, but could also prove daunting for students to program. Euclid’s algorithm, however, should be straightforward to formalise and implement.

Concepts and skills: how gcd/hcf finding fits into public key cryptography (there must be a simple way to explain this!); creating and implementing an algorithm; functions; arithmetic; numeric data types; control structure; iteration termination; variables.

* Pseudo random number generation, used among other things, to generate keys.

It is unlikely that students will be able create their own algorithm to produce random numbers. They could, however, be given an algorithm for generating pseudo-random numbers and asked to implement it in code. This could be a vehicle for teaching students about the syntax of the chosen language. Methods for generating random numbers (or seeds for a pseudo-random number generator) from hardware could also be explored.

Concepts and skills: how random numbers are used in cryptography; the nature of pseudo-random numbers; syntax; program interaction with hardware.

Robert Timmer-Arends

23 September 2012