CSC 412/512 Programming Project – Due noon, 12 December

A good way to show not only your programming skills, but also your understanding and interest in a particular topic is to create a Program Portfolio. We will not a have final in this class, but we will have a project instead, and your project will be to create a Cryptology Portfolio. A portfolio is not just a collection of work samples and sample code. It is an opportunity for you to showcase:

- Who you are
- Your work
- Your understanding of the topic
- Where you like to go next
- And maybe even what YOU really like to work with

A significant part of your portfolio will of course be your code. You will have to show code for all sections of the book that we will cover. Some things will be mandatory parts, others will be up to you to decide on what to implement or not.

I will not force you to write in a particular language, nor to write all your code in the same language. What I will force you to do is to assume that the reader is not used to compile or run code, so you will need to carefully document not only all necessary steps to run the code (compile, inputs etc), but also how use your code (input limitations, assumptions etc). You do not need to use the same language for all your code, you may switch language for different parts of the book, and even for different subsections.

Examples is not only a great way to test your code, but also an excellent way to describe your code for the user. Make use of and provide multiple examples for each code module you write.

As the purpose is to show your skill, your code of course needs to be well documented and working correctly. By well documented I mean that you should be able to go back in a year or two read your code and your documentation, and be able to understand what the code is doing, how it can be modified or even corrected.

You will also need to display that understand the topic. This means that in your documentation you need to provide enough theoretical background, that not only the reader understands the algorithm, and how it is implemented, but there should be enough information in the text you have written that you would not need to find the (or a) book. This is the tough part, as you need to write enough to understand the concept, using your own words so you don not risk any copy infringement.

How can you do this? One way is to create a webpage, present your writing on the page and allow the user to run the code on the page, or download it and run on a local machine. Another way is to create a PDF document with all your write-ups and make the code downloadable archive. The way you do this is up to you, but it is important that your intended target finds your portfolio accessible and readable. Remember that a picture is 1000 words, but it is not enough by itself, make sure to enhance it with your own words.

Classical Cryptosystems:

Mandatory modules:

- Affine Ciphers, encode and decode, letting the user provide α, β and the text (both plain and cipher text).
- Multi-alphabet cipher (Vigenère Cipher), encode and decode, letting the user provide the key and text (both plain and cipher text).
- Frequency calculation, let the user provide a text, and calculate the frequency of the different characters in the text. This is the basis for an attack.
- Affine Cipher attacks
- An attack on Vigenère Cipher, by finding the key length and the key.
- At least one of the following:
  - ADFGX cipher, encode and decode
  - Block Cipher, encode and decode
  - One-Time pads, by either Pseudo-random Bit Generation, or Linear Feedback Shift Register Sequence, encode and decode
  - Attack on Linear Feedback Shift Register
  - An Enigma simulator.

Basic Number Theory

It is fairly obvious you will need some kind of `cryptomath` library to implement many things we already have and will implement. So create a `cryptomath`-library with the following functions:

Mandatory modules:

- `gcd(a, b)` I know they exists, but it is a good exercise to fully understand the Euclidean Method, so don't use a pre-existing one.
- `extendedgcd(a, n)` So you can find solutions to equations on the form `ax+by = c` (Diophantine equations), here you probably want to return `gcd(a, b)` together with both `x` and `y`.
- `findModInverse(a, n)` More or less an implementation of the functione above, but you can take some short cuts, because here you are probably 'only' interested in solving `ax ≡ 1 (mod n)`
- At least one of the following, either as a program or as a function in your `cryptomath`:
  - Verify that `a` is a primitive root (`mod n`)
  - Find the `square roots` of `a` (`mod n`)
  - Find the inverse of matrix `M (mod n)`
  - Continued Fraction calculator; convert from CF to fraction, decimal to CF, fraction to CF and simple square roots to CF.

Data Encryption Standard:

Mandatory modules:

- Implementation of the Simplified DES-type Algorithm, with four rounds and two S-boxes.
- The Differential Cryptanalysis for Three rounds.
- At least one of the following:
    - The full 64-DES, Use the table in the book. Notice that you can create all 16 subkeys at once, and keep them as a list of keys, this makes the decryption a little easier.
      The book mentions the IP^{-1}, but it is not in the text, here is that permutation table.

      IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

      Using the following values:

      M = 0123456789ABCDEF

      K = 133457799BBCDFF1

      Should give you the following cipher text:

      C = 85E813540F0AB405
    - Differential Cryptanalysis for Four Rounds.
    - A Linear Cryptanalysis attack on DES. Just do a search and you should find lots of papers on this.

The RSA Algorithm:

Mandatory modules:

- To `cryptomath` add at least the following functions
  - `is_prime(n)`, test if a given number is prime or not. We discussed the idea in class, and we looked at three different tests.
  - `random_prime(b)`, create a random prime between 2^{b+1}-1 and 2^b-1.
- `factor(n, m)`, factor a known composite number `n` (i.e. we know it is NOT a prime), and `m` defining the method to use from the following list
  - Fermat's method
  - Pollard rho
  - Pollard p-1

  Note, you will probably want to limit the size of the number, but at the same time don't make it too small.

- `RSA`, depending on which language you will use, you might need to implement some extra libraries to allow numbers that are larger than 64-bits. Java has the `BigInteger` class, for C and/or C++ you can import the `gmp` (already installed on the lab machines) or the `ntl` library (not installed on the lab machines). These will give you the integer size needed to accommodate `p` and `q` of significant size. They will also provide functions that you might need (`modPow`, `modInverse`, `is_prime` etc), but I would recommend that you take the opportunity to fix your own `cryptomath` library to allow for arbitrary integer precision.
- Implement at least one of the following factoring methods:
  - Quadratic sieve (described in the book)
  - Shanks's square forms
  - Lenstra elliptic-curve factorization (described in the book ch. 16.3)
  - Or any other integer factorization algorithm
- Implement at least one of the following
  - Sieve of Sundaram
  - Wheel factorization

    Both are used to generate a list of prime numbers

  - Solovay–Strassen primality test
  - An `RSA` attack other than factoring (low exponent, short plain text, timing attack)

Assignment Submission:

Your project is due and should be accessible no later than noon on the day of the finals.

Grading:

Your project will be graded on three things:

• Your code, functionality correctness	40%
• Your narrative, algorithm description etc	40%
• Your code documentation	20%