

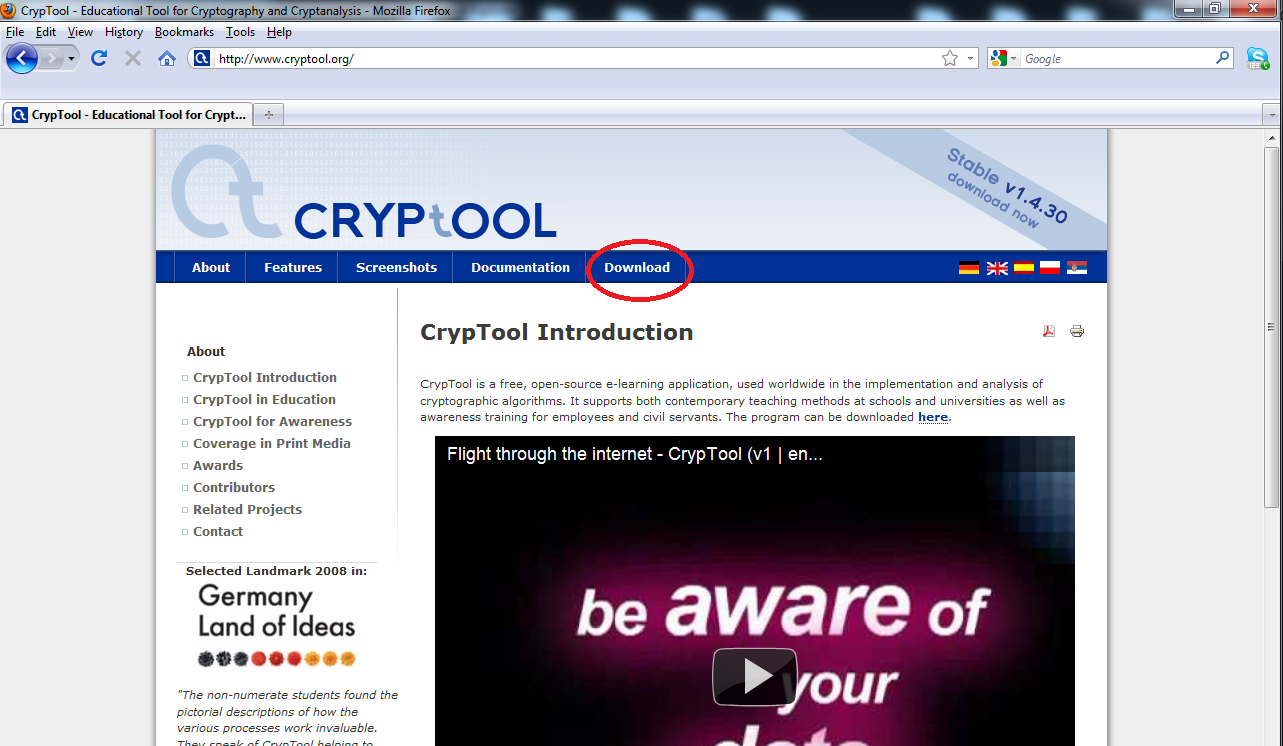
**Computer Security**

**Tutorial 5: Crypto**

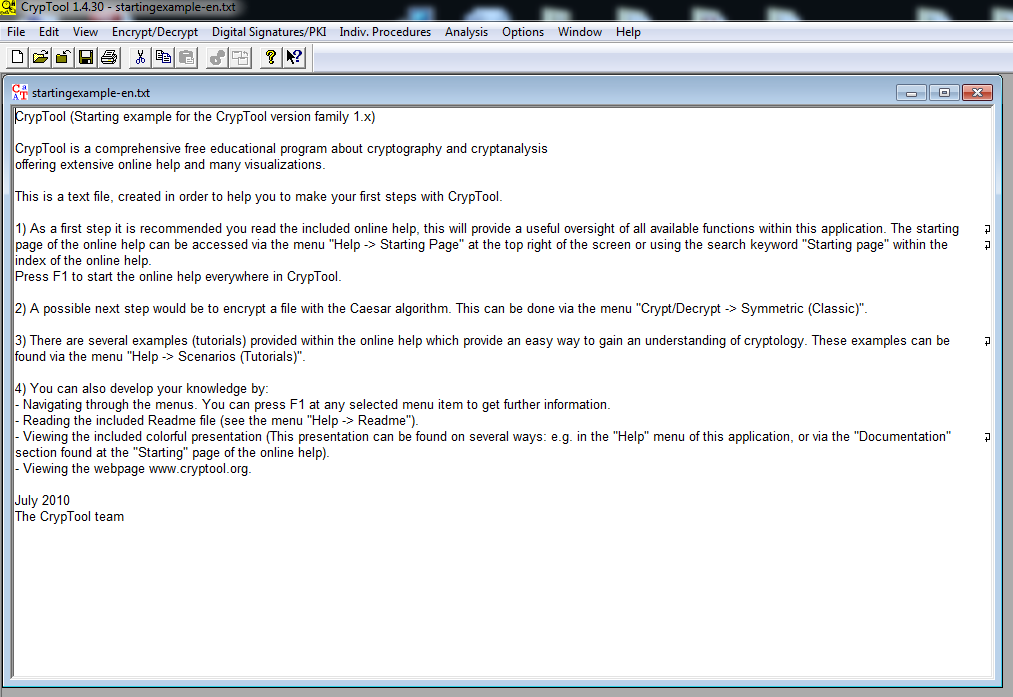
**Part 1: Cryptool**

The following activity may require you to spend more time on it than is allocated in the tutorial. You can experiment with the following education tool in your time. However, as with any downloaded code, there is a risk of malware or instability being introduced into the system. Therefore students wishing to experiment with this tool do so at their own risk. At a very minimum students should backup all important data and scan the executable with reputable, up to date anti-virus software before running the tool.

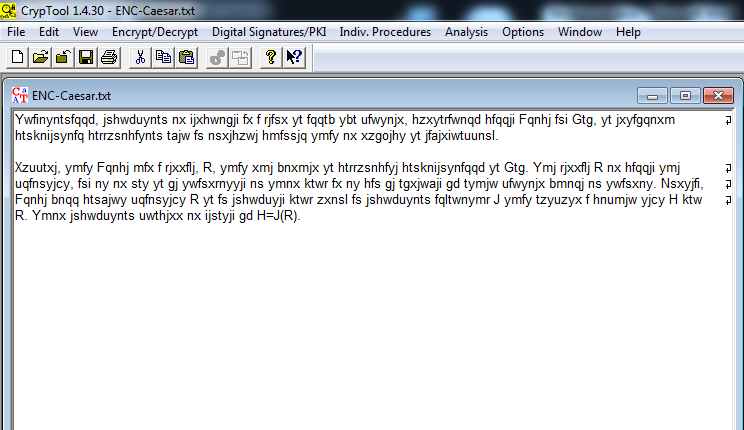
1. The tool we will be experimenting with is called CrypTool. This can be obtained from http://www.cryptool.org/ you will need to follow the prompts using the *download* link as per the diagram below.



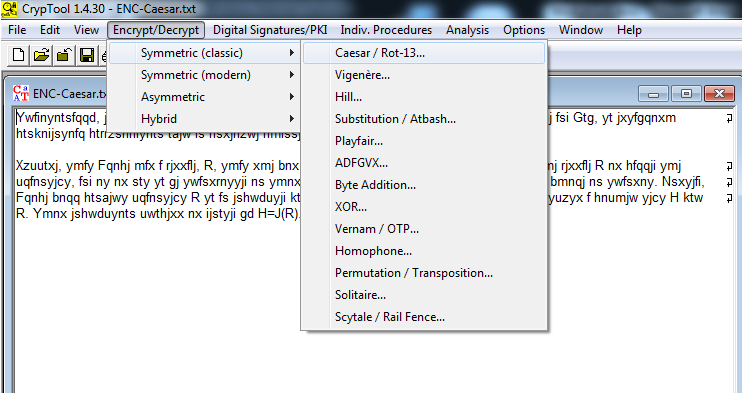
1. We will be using the current stable version which at the time of writing this is version 1.4.30 (this may frequently change due to updates). Download the software (approximately 50MB) and install the program on your Windows computer.
2. Once installed, run the CrypTool program. As per the graphic below, the menu bar encompasses a range of options which allows you to experiment with various encryption/decrytion algorithms.



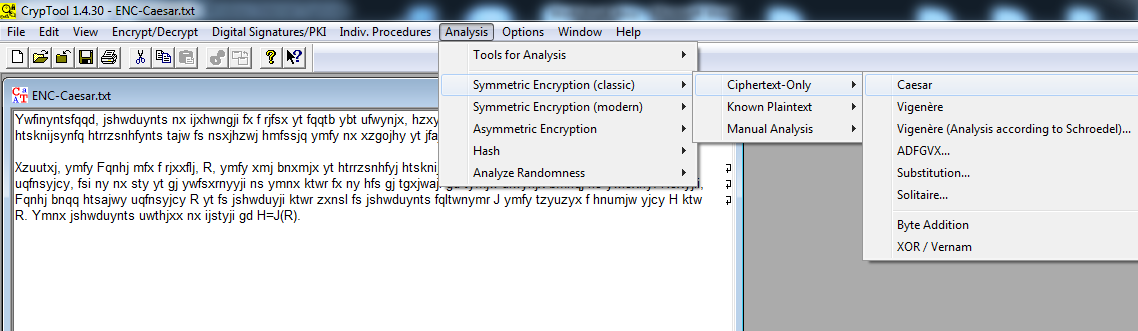
1. The default start up page contains a *startingexample-en.txt* file. We will not be using this in our tutorial so close this down by click on the *x.* The program functions by utilising the file currently open in the main program window.
2. First obtain the first encrypted file from Blackboard under this week's module, namely *enc-caesar.txt.* Save this file on to your desktop or another area which you will remember. Next using the menu options click *File-Open* then browse to the location on your computer where you saved the *enc-caesar.txt* file.
3. As per the diagram below you will notice that the text is somewhat encrypted and unreadable. *If you have the time, you may like to manually try and decrypt the message knowing that it is using a caesar shift cipher where the number of shifts is currently unknown. More information on decrypting this manually can be found by searching for manual caesar shift decryption via Google.*



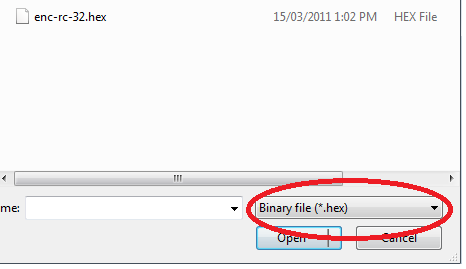
1. First we will try and decrypt the cipher-text using a manual yet automated method in this software. On the menu options click *Encrypt/Decrypt - Symmetric (classic) - Caesar / Rot-13...*as shown in the diagram below.



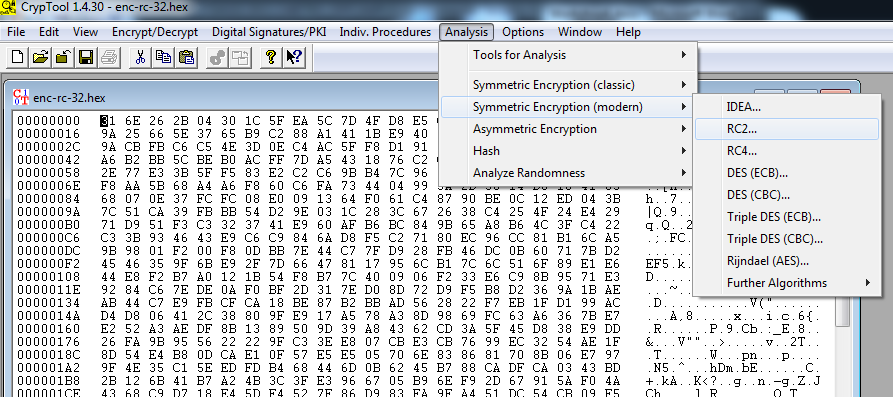
1. Remember the Caesar shift cipher works by changing the value of a letter by a specific number of shifts in the alphabet. So if the number of shifts is set to 1, then an A would become a B. The letter B would become a C, and the letter Z would start from the beginning and become an A.
2. In the window that popped up you will see that there are a range of options for you to encrypt/decrypt the cipher text. Using a process of trial and error change the *Number value* starting at 1 and click *Decrypt.* Have a look at the resultant plaintext to determine if you have entered the correct value. If the text appears to be unreadable then chances are you have entered an incorrect *Number value.* You will need to keep re-iterating this processand enter an alternative *shift value* until you can read the resultant text.
3. As you can see the process can be time consuming (even whilst using a computer) even for a very simple caesar cipher method. You should now have two windows open within CrypTool. The original cipher text, and the resultant plaintext. Close the resultant plaintext output so that you only have the cipher text window open.
4. Next we are going to attempt to brute force attack (crypt-analysis) the ciphertext *enc-caesar.txt*



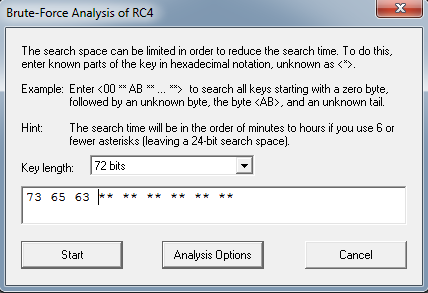
1. From the menu options click *Analysis - Symmetric Encryption (classic) - Ciphertext -only - Caesar* and follow the prompts to undertake the analysis of the cipher text.
   1. How is the cipher text-only attack being carried out?
   2. What is the advantage of using such an attack?
   3. What is the disadvantage of using such as attack?
2. As an experiment encrypt your own *larger* file, and then attempt to decrypt it using these same methods. To encrypt your own file using this approach you will need to use the *Encrypt/Decrypt* option in the menu options, however if you are going to encrypt text using the Caesar cipher algorithm rather than click *decrypt* you will now select the desired shift value followed by the *encrypt button.*
3. We are now going to experiment with something a little more sophisticated. Next download the file *enc-rc-32.hex* from this week's module. As the name of the file suggests this is a file containing hexadecimal values. If you attempt to open this file in a hexadecimal editor (hex editor) you will notice that it appears to be all garbage (encrypted).
4. In the same manner that you opened the encrypted Caesar file, open the encrypted file *enc-rc-32.hex* that you downloaded from this week's module on Blackboard. As per the image below if you cannot see the file you downloaded and tried opening, then you may need to change the file type as marked in red below. The default file type is set to *\*.txt* although for this exercise we will be using the *\*.hex* file type.



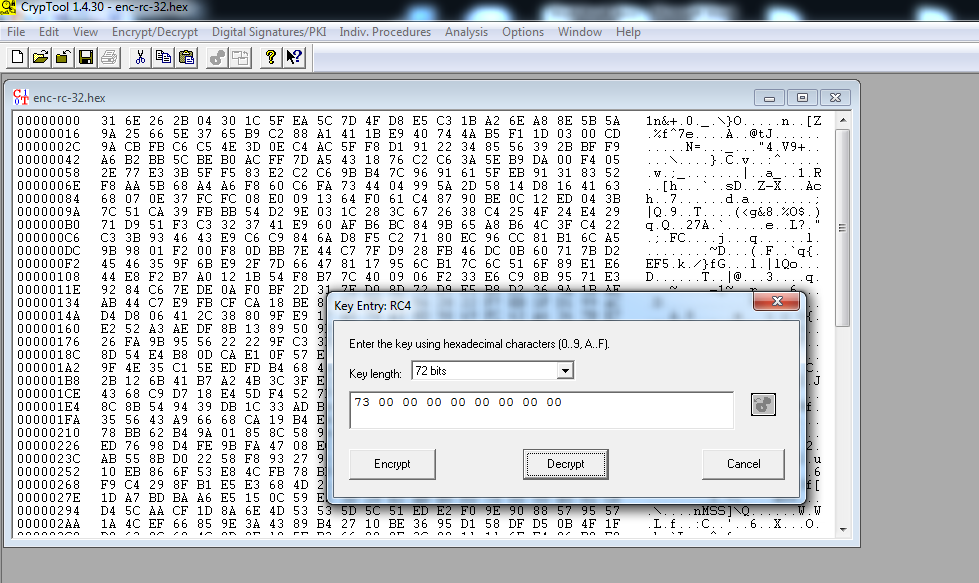
1. Let's first attempt to use a brute force attack on this encrypted file. As per the file name we know we are using an Rivest Cipher and fortunately the educational tool CrypTool supports both RC2 and RC4.
2. As per the previous exercise using the menu options head over to *Analysis - Symmetric Encryption (modern) - RC2...*



1. You should notice that it asks you for a Key Length. Unfortunately we do not know what the key length is yet. So as an experiment lets assume and try using a 128-bit key length.
   1. When you click *Start* how long does the program predict that the brute force analysis attack will take?
   2. Why is such a brute force attack simply not feasible?
   3. Think about what you could do to improve or reduce the amount of time required?
2. Since we simply do not have enough time or power to brute force attack the 128-bit RC2 key, let's click *cancel*. Obviously we want to try and decrypt it still. Hence, let's try using the RC4 option. So using the instruction from point 16, this time select RC4 and again select a 128-bit key length.
   1. When you click *Start* this time how long does the program predict that the brute force analysis attack will take?
   2. Has the amount of time required increased or decreased?
   3. Using Google or another source of information try and find some information regarding what makes RC4 different from RC2?
3. So we are back to square one and still haven't decrypted the file. Here are some tips, it is definitly using RC4 and the key length is only 72-bits. You can try using the above steps with that information in mind, but I predict that the amount of time required will be reasonable high.
4. As per the image below, the program permits us to include specific hexadecimal values to reduce the time of the key length analysis. Hence, the first 24-bits (hex) are respectively 73 65 63.



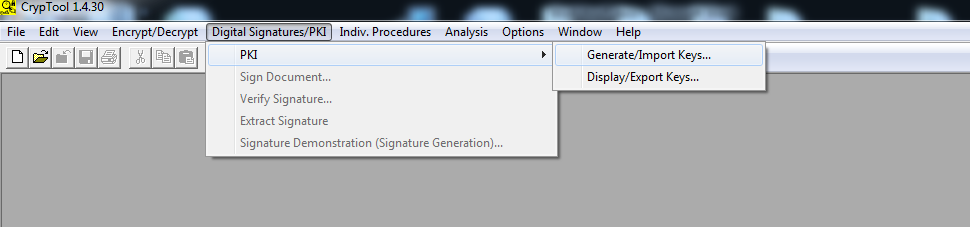
1. Click start on the program again.
   1. How much time is required to brute force attack the cipher text this time?
   2. Why has the time value changed?
   3. Have a think about what else we could do to shorten the amount of time required?
2. Click *cancel* as time does not permit us to undertake this approach. Hopefully you are beginning to see the similarities between key length and overall cipher strength.
3. So we want to completely decrypt the cipher text. The ASCII key is security+ all in lowercase. You will need to find a way to determine the hexadecimal value of security+. ASCII to hexadecimal tables and conversion programs can found using a search engine such as Google.
4. Once you have the hexadecimal equivalent, using the menu options go to *Encrypt/Decrypt* - *Symmetric (modern) - RC4...* and type in the hexadecimal value into the popup as per the image below. Once you have done this click *Decrypt* for the resultant plaintext file from the RC4 cipher text.



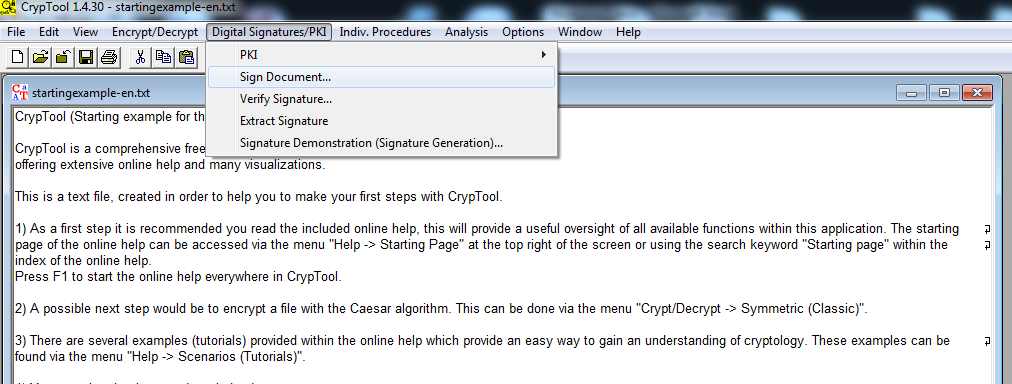
1. Completed. Feel free to play around with different encryption ciphers/algorithms and different keys.

**Part 2: Asymmetric Encryption**

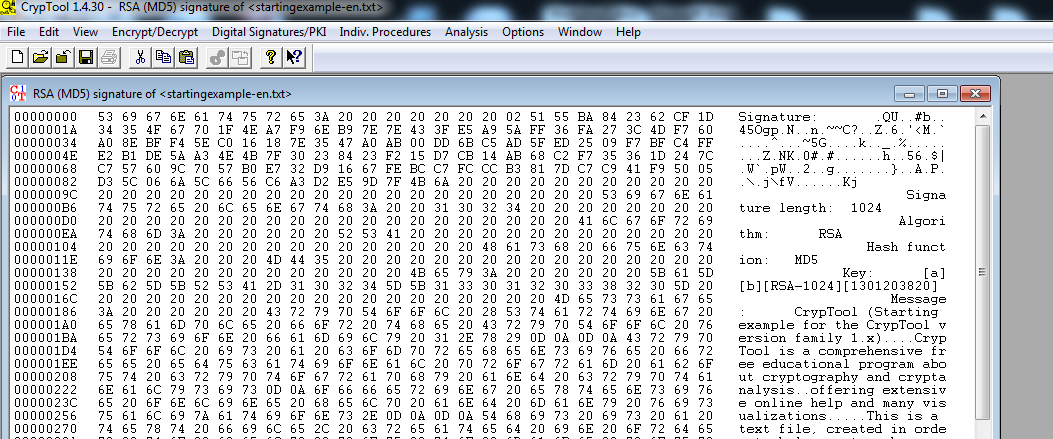
1. As per the lecture last week, Asymmetric encryption makes use of public and private keys. So first we need to generate a set of public and private keys. As per the diagram below go to *Digital Sigatures/PKI - Generate/Import Keys...*



1. As you should be quite comfortable with the CrypTool education program, you should be able to work out the next few steps. However, if you are unsure, you are basically required to fill in the required fields that the program asks for. Look at the user data that CrypTool is asking.
   1. Why is this userdata required when utilising Asymetric Encryption?
   2. Why does the program require that you move your mouse around when generating your keys?
   3. How does this process differ from using symmetric encryption?
2. Using the default *startingexample-en.txt* file as our base text, you will now attempt to encrypt the digital document. Click *Encrypt/Decrypt - Asymmetric - RSA Encryption*. Notice that although we are encrypting the document ourselves, it asks us to choose the recipient.
   1. Why does the encryption component ask us to choose our recipient (think back to symmetric versus asymmetric encryption)
   2. Which key (private or public) is being used to encrypt the document and why?
3. Once you have encrypted the digital document successfully, you should notice a new file has been created with lots of hexadecimal outputs.
   1. If you have the opportunity, compare your output with another student in the class. Will the cipher-text be the same or different?
   2. Imagine you accidently forgot the pin, or have deleted your key pairs. As a result you decide to generate a new private and public key pair. If you use the same userdata as you did the first time, will the cipher-text be the same or different (you may like to try and experiment with this)?
4. Next we are going to attempt to sign our document (digitally). Remember, when you sign a document you do this via your private key (because you are the only one who should have access to this).
   1. Why aim of security does signing a document ensure?
5. Close any cipher-text windows that you have open, leaving open only the original *startingexample-en.txt* file/window. As per the diagram below click *Digital Signatures/PKI - Sign Document...* You will notice that it asks you to select a hash function and the private key that you wish to use.



1. Following the prompts, digitally sign your document. As per the diagram below, look at the right hand side column and attempt to read the type of information contained in this signature.
   1. What type of information is now contained in this document?
   2. How is digitally signing a document beneficial?



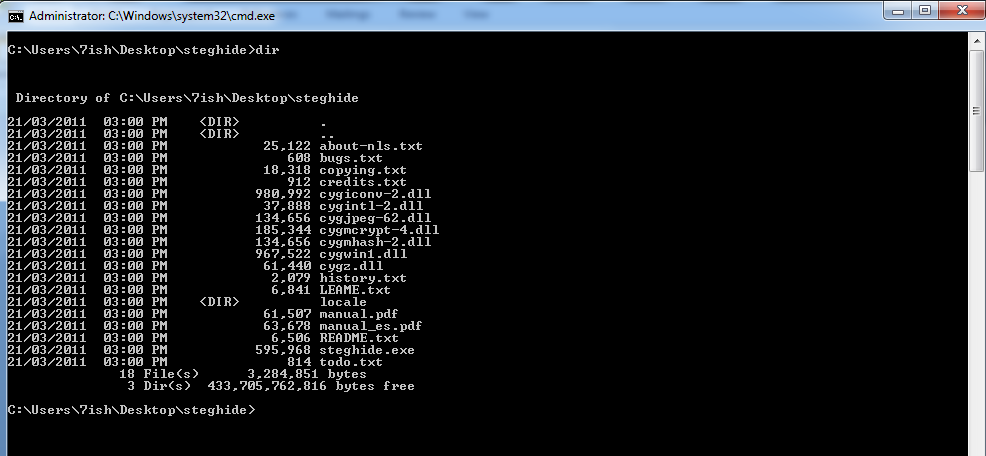
1. At this point you may be a little confused as to how the signature process occured. Fortunately, the CrypTool program has a nice tutorial feature to demonstrate the process to you. Close the signature window in CrypTool, so that you only have the original text window open. Next click on *Digital Signatures/PKI - Signature Demonstration.* From here on you can click on the shapes following the prompts to have a live demonstration of digitally signing a document.
2. *Once completed continue onto Part 2, which will not utilise a different program to demonstrate encryption.*

**Part 3: Steganography**

Look at the two steganography pictures under this week's modele on blackboard. The pictures look identifical, except the second picture contains an embedded file. Can you see any differences in the pictures? You may need to use photography software other than Microsoft Paint. Verify your decision using a hash digest of the two files via a program such as HashCalc (http://www.slavasoft.com/hashcalc/index.htm)

Last week we looked at changing the meaning of text (encrypting) resulting in a ciphertext that not everyone can read. Sometimes it is more practical to hide a document, image, or even evidence within another file.

1. You can either obtain *Steghide* from this week's module on Blackboard, or download it directly from http://steghide.sourceforge.net/
2. Extract the contents of the zip file into a location you will remember. If you are running this program from the lab machines it is advisable to extract the contents onto your desktop due to the permission restrictions.
3. Under this week's resources there are two files. The first file is *StatueOriginal.jpg* and the second is *StatueSteg.jpg*. Save both files to your steghide folder. This will be the same place where you extracted the steghide program from point 2.
4. To familiarise yourself with how the program functions, it is advisable to briefly read through the manual provided with the program. In the steghide folder you will notice a *manual.pdf* file which contains the list of commands to extract and embed documents into picture files. Once you feel you know how to extract files proceed to the following points.
5. Open *Command Prompt* and navigate to the directory where you extracted the steghide program. If you are not familar with using the Microsoft Windows based *Command Prompt* then you make like to refer to the *CommandPrompt.pdf* document located in this week's resources. This document provides you with an example (not related to this unit) of the basic functionality needed to navigate between directories.
6. Once *Command Prompt* is open, navigate to the steghide folder. As per the image below, if you have navigated successfully to the correct directory and listed the contents of the steghide folder, you should see a range of program files. Please note, that the directory path will be different for each person.



1. Using the steghide manual, attempt to extract the file from the StatueSteg.jpg file. The password required to decrypt (and extract) the file from the image is *test1*
2. Once you havesuccessfully extracted the image. You may like to try and play around with embedding a document into an alternative image.
   1. What is the maximum allowable size of files compared to the image?
   2. Are there any notable differences between the two statue images from blackboard?
   3. If you ran both files through a hashing program (i.e. MD5) would the resultant digests be the same or different?

**Part 3: Distributed.net**

Have a look at www.distributed.net

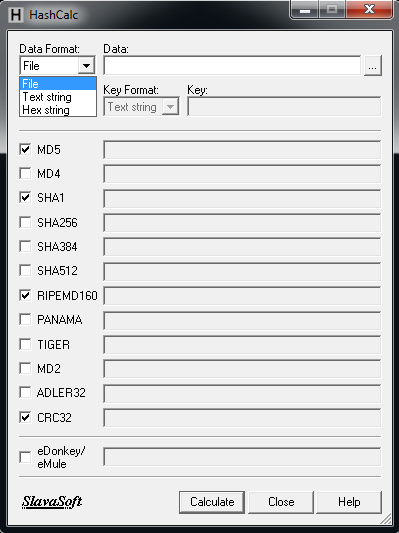
Specifically, have a look at the stats for the on-going RC5-72 project and answer the following questions?

* How many keys would this project have to check, given a worst-case scenario?
* What is the key size of RC5-72?
* How many keys would this project have to check, given a best-case scenario?
* How many keys have they currently checked?
* What percentage of the keyspace is this?
* How long has the project been going?
* How long do they estimate that it will take to have a 100% probability of finding the key?

**Part 4: HashCalc**

To this point, you may be wondering why it is important to attach a hash value/digest to blackboard when submitting your assignment. The following small exercise should (*hopefully*) answer this question. In part 2, you may have already downloaded and installed HashCalc. If not go to (http://www.slavasoft.com/hashcalc/index.htm).

1. Firstly open notepad on your computer. You are required to create and save two seperate files on your desktop file1.txt and file2.txt
2. Open file1.txt and write the string "This is file 1" and save the document. Then open file2.txt write the string "This is file 2" and then save the document.
3. Next open the HashCalc program. As per the diagram below, you will see that we can input any data format, and choose any specific hash function that we want. For this exercise our data format will be a *file*, and we are predominantly interested in using *MD5*.



1. Drag and drop file1.txt into the data field in HashCalc. Take a note of the MD5 value that is generated as a result of this particular file. This MD5 value is the mathematical output as a result of this specific digital document. Note down the MD5 value somewhere.
2. Next drag and drop the file2.txt document into the data field. Observe and document the MD5 value now.
   1. Has the MD5 value changed?
   2. Why did this particular process occur?
3. If you did not write down the number (or at least a part of it), go back and note down the value of each file. Next, delete file1.txt so that you are only left with file2.txt
4. Now, rename file2.txt to file1.txt. Once this is done, drag and drop file1.txt (which is actually file2.txt renamed) back into HashCalc.
   1. Does the MD5 value remain the same or different?
   2. Which file is the MD5 value corresponding to?
5. Next open the file again, and change the string "This is file 2" into "This is file 1" and again drag and drop the file into HashCalc
   1. Has the MD5 value changed?
   2. Does it correspond to a particular MD5 value obtained previously?
   3. What changes need to be made to the file for it to alter the resultant MD5 value

Lastly, for this week, since you now have a little experience with generating hash values, create a list of when and where it would be beneficial to use a hash function/program in computing and security. Which aim of security do hash functions ensure? How would has functions be beneficial in the field of digital forensics?