Nate T Rouse

HSFC

Encryption engines

hashes, binomial, and a self-developed engine

# Problem identification

## Problem:

I am developing an all in one software for encrypting data, it will include a Hash algorithm, a polymorphic engine and a binomial encryption engine, all designed by myself.

## Types:

The Hash algorithm will output a hexadecimal hash of 128 bits, no matter the size of the text inputted.

The polymorphic engine uses a key of 4 hex characters to augment an equation, and change the values of the equation to a certain amount defined by the last digit of the key.

The binomial engine will use the Pascal’s triangle to develop a multiplier which is then decreased in size by a logarithmic equation which will decrease the values outputted from the binomial choosing function by a scale that increases the bigger the value gets.

## What it has to do:

The engines will take in plain text and output Unicode characters so that only the programme itself can interpret the data making it harder for a single person to crack the algorithms. It will also have a function to output the text as a text file which can be saved to the computer itself. It will also have a feature to email the encrypted text to a specified email address and will check that the email address follows the convention of an allowed email address. It will be used by anyone that wants to keep their conversations secret, or their files safe.

## Computation:

The programme will use loops that are defined as the length of a string, and the item they are evaluating in order to keep the code finite, and no code will be left to infinitely loop therefore not using up the systems resources indefinitely.

The instructions will be made as efficient as I am able to make them to speed up the process, but the binomial functions are very slow in any environment so will take time therefore they will all have a progress bar to show how much of the text has been encrypted showing the user that something is happening in the background.

The algorithms will work up to the maximum size of a string, therefore the data handling of the system will fail before the algorithms do (2 billion characters).

The algorithm will halt when all of the characters have been decided and have been appended to an output string which is returned to the user in one of three ways (output to a text box, email to your email, and save the file in a location) this will be monitored by a timed code to check it progress as stated before, if the programme does not progress the user will be notified after 10 seconds of idling if the algorithm has not finished.

The Memory is only limited to the space physically in the machine, but there will be CPU optimisations made to speed up challenging operations.

### Reality:

This solution will simplify reality by processing large numbers that a normal calculator or human can do, to safely protect information, the main details in this is that everything encrypted is relative to its key, if you do not have the key, the message can’t be read as the information is hidden by a set of algorithms, reality will be omitted in the fact that some of the numbers being computed are insanely huge, and will not be calculated to their definite value but their approximations using different methods to provide accuracy as well as consistency.

Instead of doing a complex sum consisting of thousands of steps to compute: n!

I will be using which is called the Stirling approximation.

This provides an estimate to the value of the factorial which gets closer to the curve of n! As n gets bigger, it is also very accurate only being a few decimals out, therefore I can use the Math.Ceiling function to get the actual value.

The algorithms will be written In VB therefore allowing me easy access to the streamreader, and writer function to fulfil one of my requirements, this is the same for the email system as well, using the system.net.mail library.

The algorithms will be given a key for the ones that require a key, and an input string with the users’ data, after the programme has altered each character individually, it will output the finished product, and the program is restricted by the length of the string so no indefinite loops will be created.

When the user confirms their selection, they will then have to wait for the algorithm, if the key is 0000000000000000 for the binomial encryption, a defined key will be used, as there is no way to encrypt a message using 0000000000000000. With all the other types the key of 16 0’s will work due to the algorithms.

The algorithm will work with strings up to 2 billion characters long. The integers can handle up to 2e308, therefore my algorithms will be written to keep numbers below this.

As stated above after encoding every character the program will halt and output the correct output. This will happen because after every character is augmented the algorithm appends it to the output string.

The algorithm will be optimised to use the most of the cpu it can, therefore speeding up the programme, the programmes resources will be limited by the physical hardware in the computer it is ran on, therefore the algorithm has no limit to the memory used.

These algorithms are engineered to omit a person from being able to understand a message, and therefore omitting the reality of their language from the outputted result, protecting their data.

The only detail that is important is that all of the algorithms are different, but also the same, the way in which they are obeyed is the same, but every time it is given a different key, It will change as the parameters are different, therefore no 2 of the encryptions are the same, as some of them also take into account the strings length as another factor of obscurity.

The system will have to be connected to the internet, with port 587 not being blocked for the emailing system to work. The algorithm requires only a string and a key to be inputted, with a button press as well to initiate the algorithm.

My programme will be broken down into multiple functions that will be global, so that they can be stored in the same class, and called whenever by referencing the variables to be used in the algorithm. This will therefore cut down on the amount of code required to be written as everything will be decomposed in to repeatable chunks.

My algorithms are based on repetition as they execute the same algorithm on each character in the inputted string, therefore requiring the careful development of the loops, to ensure they are finite.

Some methods will be calculated at the same time, as then they will require less time, but will use more of the systems resources. I could split the strings into 2, and encrypt them at the same time, therefore reducing the time required to encrypt the entire thing.

### Stakeholders:

#### Theo Johnson:

Theo is a fellow computer science student, and would like to use my programme to protect some of his data on his computer, therefore protecting his information from unwanted eyes. He wants to use this to ensure that when he sends emails to his parents, or boss, that no one can see what’s in the body, if he’s sending personal data, such as bank info, or what he is doing at the weekend. He will also use it for research as he is curious as to how encryption works and hopes to develop something like I have some day.

#### Heron Lewis:

Heron is much like Theo, in the fact he is also interested in encryption techniques, he also is very interested in encryption as he hopes to go into a future of computer science, and maybe do a degree in cyber security, therefore allowing him to use my engines as a test of his decryption methods when he develops them, he also wishes to use this for fun, to try and decrypt the strings by hand, putting his brain at work, especially in the binomial one where massive numbers are used to gain a diverse boundary of values.

# Research

## Binomial

### What does binomial mean?

An algebraic expression of the sum or the difference of two terms.

Therefore using the formula provided for binomial expansion, this allows us to implement this into an algorithm and make use of the fact that the equation is reflective, therefore being relatively unpredictable to add a further depth of encryption as somewhere in the string, the algorithm starts to decrease instead of just continuing on the same trend.

KEY COMPOSITION:

AA BB CC DD EE FF GG HH

N b c g f h u a

N: represents n in the formula

B: represents b in the formula

C: represents c in the formula

G: represents the period that x will go to, starting from h’s value

F: represents the log base (shown in picture is the natural log)

H: refer to g

U: is the characters added to the beginning

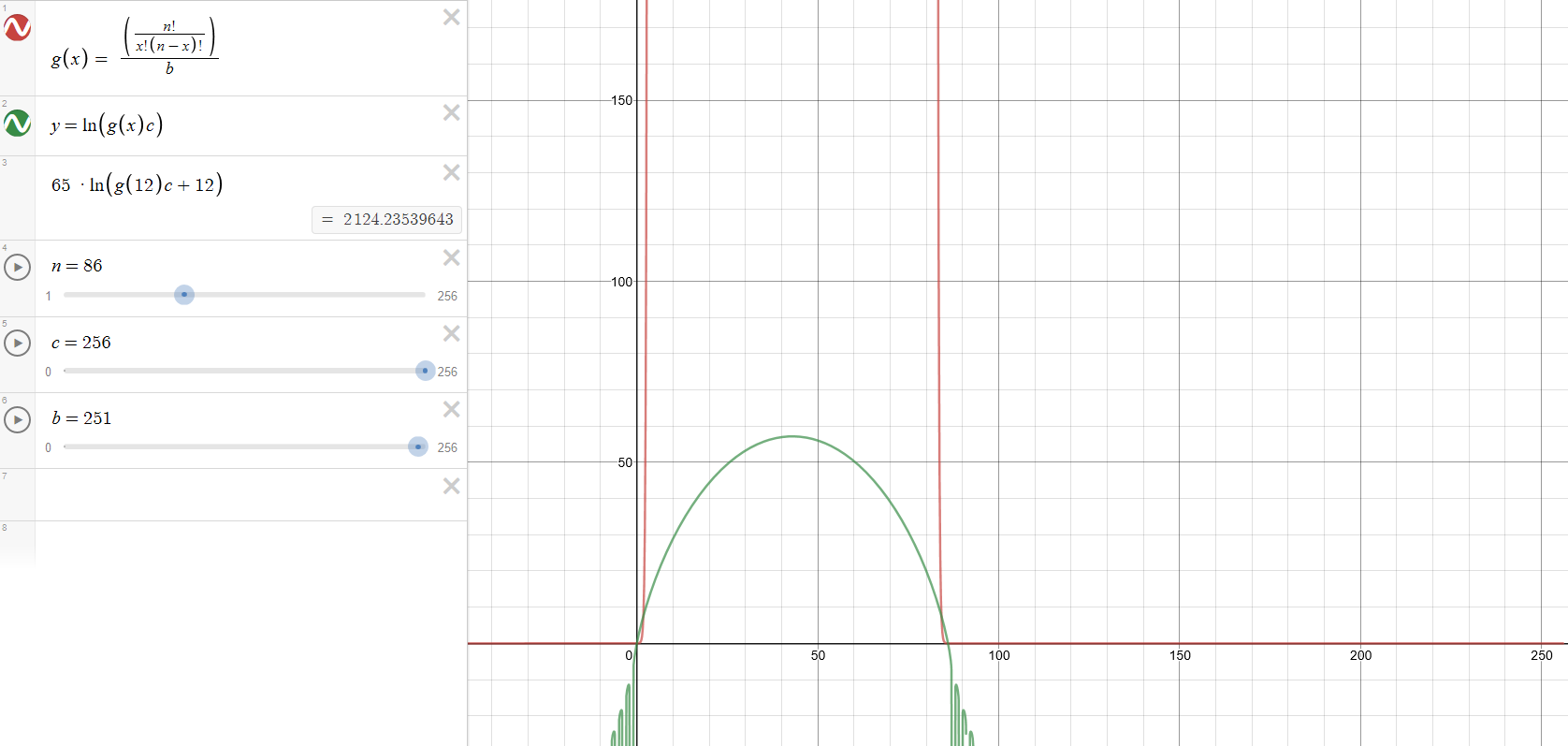
A: is the characters added to the end

These are the formulas that are going to be used

Binomial formula:

The binomial formula augmented by b, to reduce the size of the elements

Concatenating logarithm:



This graph shows how the formulas alter the values, the small green curve is the logarithm of the binomial expansion which is the red curve which reaches quite high up.

This is my own design, and I can’t find any other uses of this algorithm to encrypt any data.

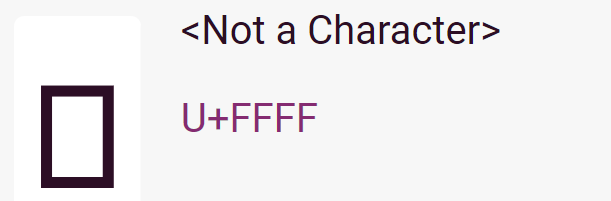
The thought behind the method is that we bind the encryption to the binomial distribution of its elements. To do this I will be taking the first 4 terms of the key and use them in the formula, and x will represent a value in a step loop to randomly encrypt each letter with a different part of the binomial curve as shown above. The step loop is determined by starting at g and stepping up to g and then back again. This term of will determine what the multiplier of the Unicode value is.

This method is very secure as the one key can create a large range of outputs due to the curve which provides a large range of spread. If I was to get a return of my algorithm that my multiplier was 7.85, at x=4 and n=20 and b = 2, my value for A would be, 65\*8.75=568.75, this would convert A to: ȹ

Then on another value of x the multiplier can change to 15.8 this would make A’s value shift to

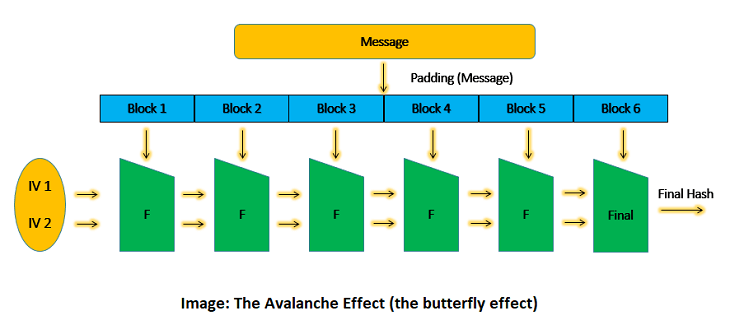
65\*15.8 = 10271, that means A converts to: ⠟.

Another good reason that this encryption works well is that some times the formula can return characters that cant be interpreted because they are indescribable, and some Unicode characters return a box which has many values assigned to it. This is because the default character for any unassigned value is the same box, but only a computer can interpret the difference.

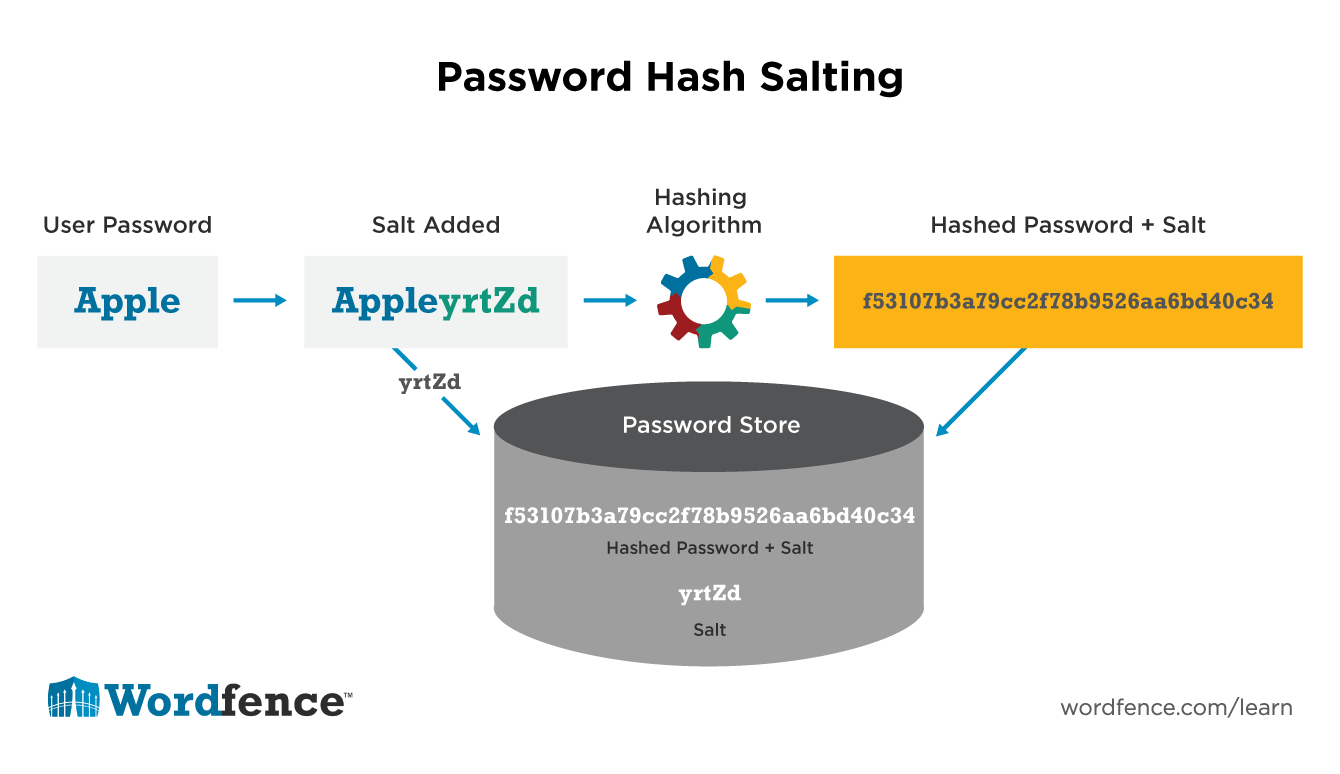


## Hash:

I’m going to use the butterfly effect, this takes a string, sorts it into 128 bounds in which then are ran through the hash algorithm separately, and then it returns a 128 bit hash.



It works like this, the message of how many bits long, the message is split up equally between 128 data blocks. The first block is ran through the hashing algorithm and then fed into the next block as a hex value, if the string is only 128 bits long, each block gets 1 bit and the code runs once. Thus, making the final output the combined value of all the blocks. If you change one bit anywhere in the message, the entire hash value changes. This is called ‘the avalanche effect’.

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=2ahUKEwjixu_Mn4LjAhXYAmMBHek4BOIQjRx6BAgBEAU&url=https%3A%2F%2Fwww.wordfence.com%2Flearn%2Fhow-passwords-work-and-cracking-passwords%2F&psig=AOvVaw1UNp3AQ5aWqID_oos-mwbY&ust=1561470097749667)

##### Birthday paradox:

The birthday paradox is derived from the understanding that in n amount of people, inevitably some number of them will share the same birthday;

With all security we can assume there’s always a way to manipulate it, with hashes, the manipulation is the birthday paradox, in a 128-bit hashset, there is 16128 possibilities, but many hash algorithms will produce a duplicate of the hash at least once for every hash, this means that we can now cut the possibilities down to 1664 possibilities, which makes cracking a hash easier than the full value, this not only lets us test to see what hash values are returned, but if we find one of the birthday hashes, then we can gain access to a system with another password, with our birthday hash.

There is ways to stop this, by using a 512-bit hash algorithm, but cutting it down to 256, or 128-bit hashes, even though you only now have 256/128 bits, the hash is stronger as it was made from the main body of the 512 bit hash, therefore even though there are 128 bits, there are still 16256 possibilities if we use the birthday paradox, to try and find a hash collision.

The basis of a good hashing algorithm is derived from the algorithm that is used, as it has to be easily computable in one direction, but not in the reverse direction. Therefore, I am going to use a sum equation that has roots, powers and utilises the sine function as these functions when combined in a sum and a power, are very hard to reverse as there is not an easy way to compute the arcsine of sin2. This combined with adding the previous value to the next, will ensure that each value of hashing will be unique in the fact that that value wont be able to be reached by a string of the same length, but only greater length therefore the birthday paradox is reduced in efficiency.

##### Vulnerability index to Hash Chain Attacks:

As I am creating a hash chain using the butterfly method, it still has weaknesses, as it can also increase collisions (when two values of input give the same output hash). A way of analysing how strong a hash function is we use a probabilistic algorithm to produce a vulnerability index which dictates how strong the hash function is relative to:

* MD5
* SHA1
* RIPEMD128
* RIPEMD160

It then gives a value, the lower the value, the more secure it is against probabilistic attacks. This also means that the generation of the algorithm is better with a lower score.

##### Collision attacks:

Collision attacks are when a computer tries to force a hash algorithm to produce two hashes with a different input but the same output (nicknamed a collision)

This is typically written as hash(m1) = hash(m2) or hash(p1 // m1) = hash (p2 // m2)

Where m1 and m2 are the inputs, and p1 and p2 are appendages that are added if the two hashes have very similar values, and // denotes a formula that is used to append the two string in such a way to not change the hash value too drastically.

Collision attacks can be improved using cryptanalysis, which allows us to develop a collision attack which will always be faster than the birthday paradox previously mentioned, and then the hash algorithm is deemed ‘broken’ and is then slowly discontinued with a more secure version being used in its place, this also means that hash libraries containing passwords have to be regenerated, and therefore services will often force a password change as the old ones are no longer secure.

Common examples of collision attacks being successful is that of the MD5 variant in which a typical computer can find a collision in a few seconds, whereas when the attack is being developed it can take a supercomputer several hundred or thousand hours to produce a collision, in which cryptographers can then use to break the hashing algorithm.

Collision attacks are very dangerous in some cases, as a document can be evaluated to receive a certificate, and then be proven as safe to install, but a document that has malicious content with the same hash can then have the certificate copied over creating a malicious file, that the authorities deem as safe.

##### Chosen-prefix collision attack:

This is specific to the Merkle-Damgård hash functions. In this case the attacker choses to append different calculated values in order to result in whole documents having equal hash values. **This attack is much more powerful than typical collision attacks.**  This is because it can then be exploited in the same way as above and can embed malicious text into a ‘safe’ environment and collect data undetected as it holds the same value as an ‘inert’ text file.

To show the increased potency, in 2007 this method evaluated MD5 and found an attack in 250 evaluations, whereas the birthday paradox would require 264 evaluations to find only one set of values that can be exploited, whereas this method provides a ‘algorithm’ to find further collisions. This attack was used to copy a certificate from one registered domain to one that has never been registered and therefore impersonated a ssl secured website to insert malware onto visitors. Another example of this was the flame malware which spoofed a certificate owned by Microsoft on one of their root components that still used MD5 to hide malware inside the systems root controller.

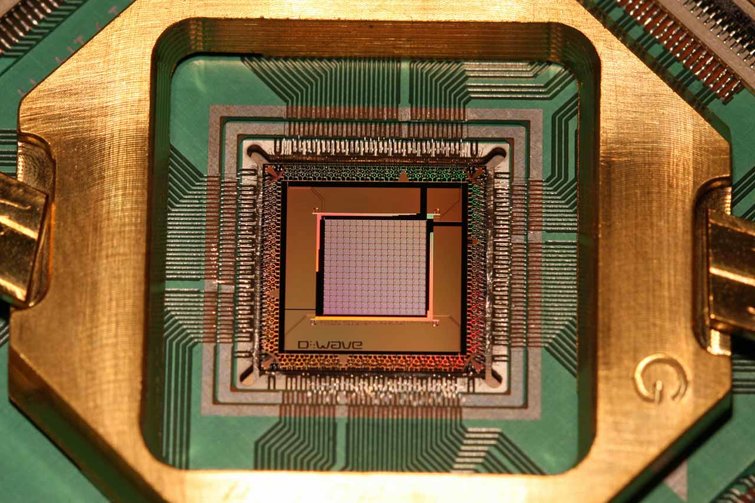
This year cryptographers found a chosen prefix attack against SHA-1, the most commonly used construct, taking between 266.9 and 266.4 evaluations to define a collision, and costing less than 100,000 us dollars [source : <http://en.wikipedia.org/wiki/Collision_attack>]

##### The Universal Hack:

Quantum computers provide a gateway to creating the universal ‘hack’ this means that given any set of data, that has been encrypted and a few know characters, it can work out the formula, or the input that created the encrypted text.

###### How? :

Quantum computers are unlike our ‘normal’ computers in the fact that they don’t compute sequentially, they consider all possibilities **simultaneously.** That means that as I stated before with the birthday paradox it take 2n/2 iterations to find a collision that would allow us to break the formula. A quantum computer will take nanoseconds to analyse millions of these possibilities simultaneously, and then it will compare the results using a series of quantum gates and blocks to see which sequence of bits provides a collision or the decryption key as shown above. When the cryptographers complete their tests this will break the internet as there will no longer be any secure encryption, as it will take only seconds for a q computer to break encryption that would’ve taken a computer millions of computing hours to compute half of the sequences!

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=2ahUKEwiykvLxmoLjAhWozoUKHTsNA-oQjRx6BAgBEAU&url=https%3A%2F%2Fwww.tomshardware.com%2Fnews%2Fd-wave-2000q-quantum-processor-computer%2C39347.html&psig=AOvVaw2I0dOQLkxnmLLP-LDI0Qg5&ust=1561468843612301)This QPU pictured below is the only thing needed to compute this, along with the qbit interpretation gates and the cooling kit. But this qpu the size of the average cpu, and compute this immense amount of data in a very short time.

### N3Ncrypt:

I personally developed this method. It uses a 5 bit hex key that has 165 possibilities, this means that to guess my encryption would take 1.05 million iterations to calculate my key from the output.

The way my code works it takes a series of the key characters and uses one to define a key loop which changes the augmentation algorithm for every single character. Then 3 of the digits are used as a starting augmentation to encrypt the first letters in the iterative cycle. This is similar to polymorphic encryption but doesn’t use a algorithm that changes its parameters.

It works with this code:

*If direction = True Then*

*a(1) = a(0)*

*b(1) = b(0)*

*c(1) = c(0)*

*temp = 0*

*For i = 0 To message.Length - 1*

*temp = Asc(message.Substring(i, 1))*

*temp += a(1)*

*temp \*= b(1)*

*temp -= c(1)*

*done += ChrW(temp)*

*If j < p Then*

*a(1) += 5*

*b(1) += 1*

*c(1) += 2*

*j += 1*

*ElseIf j = p Then*

*a(1) = a(0)*

*b(1) = b(0)*

*c(1) = c(0)*

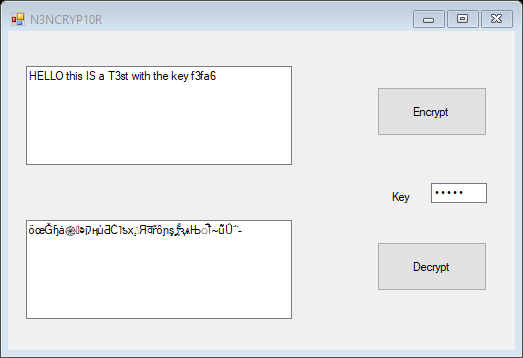
*j = 0*

*End If*

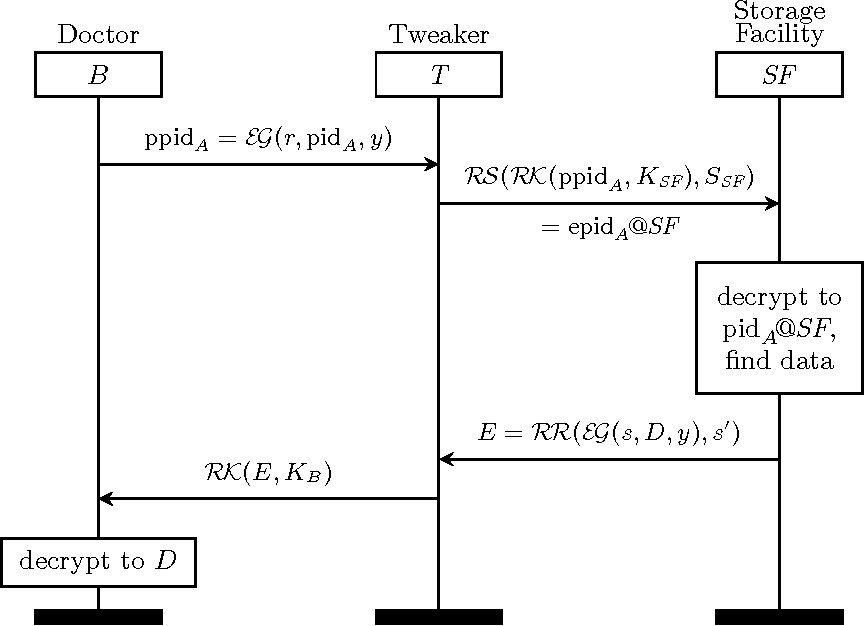
*Next*

*Return done*

It then returns the full text but it wont always be interpretable, and is best to be kept on a computer in a word processor that uses Unicode.

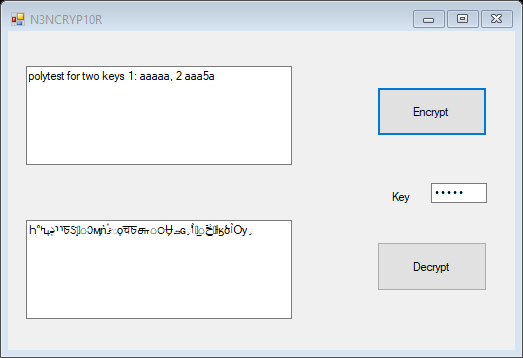


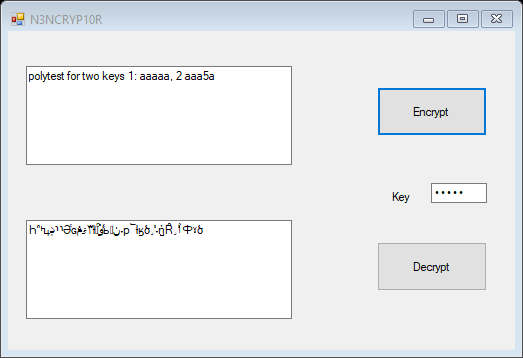
This is a demonstration of N3NCRYPT where it returns a bunch of text that can not be interpreted, and therefore is very safe against human attacks. But not computer attacks as the key is so short compared to modern day encryption as this represents a 20bit binary key, whereas newer types use the equivalent of 256 bit, which correlates to 1x10^71 times more possibilities. Therefore this I a good proof of concept, but not a good demonstration of how good encryption can be.

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=2ahUKEwiv8q_YoYLjAhUkx4UKHVgCD_0QjRx6BAgBEAU&url=https%3A%2F%2Fwww.semanticscholar.org%2Fpaper%2FPolymorphic-Encryption-and-Pseudonymisation-for-Verheul-Jacobs%2F7dfce578644bc101ae4ffcd0184d2227c6d07809%2Ffigure%2F4&psig=AOvVaw2hdSwSEx7_XX51kxhT2Qjn&ust=1561470669156527)

This diagram describes how the algorithm works, the doctor is the key changer, and the tweaker changes the way that the algorithm that interprets the key. The storage facility is where the system determines from the previous value what the output will be.

This can be shown by changing only one of the keys on an encryption.:

 AAAAA

 this is AAA5A which has changed all of the characters by just changing one of the characters, which shows how good polymorphic encryption can be, the only vulnerability is brute force attacks, as no one character translate to a single other, therefore occurrence analysis cant be done to determine which characters are encrypted in to others.