A Study of Effectiveness of Historic and Modern Encryption Methods

Author: Ryan Sadler

Supervisor: Mr. David Lightfoot

Table of Contents

Glossary

AES	–	Advanced Encryption Standard, the internets “default” encryption algorithm
Asymmetric	–	Two separate keys are used, on for encryption and one for decryption
Block Cipher	–	A cipher that encodes pieces (blocks) of a plaintext at a time
Ciphertext	–	The result of encrypting the plaintext with an algorithm
DES	–	Data Encryption Standard, an outdated algorithm, widely replaced by AES
Encryption Ratio	–	The difference in size between the plaintext and the ciphertext
IoT	–	Internet of Things
Plaintext	–	The message, or data, to be encrypted
Polyalphabetic	–	Contains multiple alphabets
Public-Key	–	A version of asymmetric key systems where one key is broadcast at the start of communication, and one is kept private
RSA	–	Rivest-Shamir-Adleman, a public-key modern encryption algorithm
Symmetric	–	The same key is used to encrypt and decrypt
Triple DES	–	A variant of DES where 2 or 3 keys are used in a sequence of subsequent encryptions and decryptions (also known as 3DES)

Abstract

This report focuses on a gap I found in research based on what encryption algorithm is most suited for encrypting text. Therefore, I have designed my own version of 10 different encryption algorithms; 5 historic and 5 modern and tested them against this task. I have implemented them using Python 3.12 and included a basic GUI for users to recreate my results and test their own theories. By the end of this report, I will have a detailed report on the performance of these separate algorithms for each separate task, scored on a scale I have come up with, including but not limited to Encryption Speed, Decryption Speed and Encryption Ratio.

Acknowledgements

I would like to thank Mr. David Lightfoot for assisting me throughout this process.

Introduction

Background

Very frequently, when I am creating something that requires security, I have wondered what particular algorithm would be best for the use case I require, be it encrypting text to send over a network connection, or simply encrypting a file to store. This is why I have decided to complete this study, as a means to be a easy access for others who require the same information.

Aim

This study will aim to judge how effective different encryption methods are for different purposes. I will be testing 10 separate algorithms on both text and image encryption. These will be 5 historic algorithms and 5 modern algorithms as detailed below. For the image encryption, instead of modifying the algorithms themselves, I will be converting the binary data stored inside the PNG files into text based on a fixed alphabet that I have created, similar to the base64 alphabet but with the + and / symbols replaced.

Objectives

My objectives are as follows:

- To develop a scoring system to judge effectiveness of algorithms
- To research and develop 10 separate encryption algorithms
- To develop software for testing the encryption methods using Python's inbuilt tkinter module
- To use this to score the algorithms on both image and text encryption
- To bring this data together into an easy-to-read format for other developers to use

Product Overview

My end product is this report that will bring together all the data I have collected together for other developers to use when they need it. The final algorithms I have chosen will be: Caesar Shift, Substitution Cipher, Vigenère Cipher, Rail-Fence Cipher, Enigma, RSA, DES, Triple DES, Blowfish and AES. The scoring system will be based on encryption and decryption speeds, key size, encryption ratio and key generation time. I will also include an additional analysis of each algorithm based on security and how easy it is to crack.

Background Review

I found a pre-existing study around this idea *(Panda, 2016, pp.278-284)* that evaluates various algorithms using Encryption time, Decryption Time and Throughput. This differs to mine as I intend to not only use audio encryption, but I will also be judging the algorithms differently. They have a good range of modern encryption algorithms, though no historic ones and the data they gathered is very clear and easy to read. I can use this data to double check my code in my testing stage using the same inputs they did.

Another paper I found *(Yadav and Majare, 2016, pp. 70-73)* evaluates them very differently (Throughput, Key Size, Encryption and Decryption Speed and Time, Encryption Ratio and Level of Security Issues) however they do not encrypt images or audio. I believe this is a much more robust way of judging algorithms than just purely speed and I used this as a basis for how I judge my algorithms. A very recent paper I found *(Paradesi Priyanka et al., 2022, pp.295-300)* does things differently. Rather than scoring them they have compared them using the standard speed, security and power consumption most of the others use, however they also compare the against each other with their age, algorithm type (symmetric/asymmetric) and inherent vulnerabilities (brute-force, side-channel attack and oracle attack). I like this approach as you can truly tell an algorithm’s strength and security with age.

An older paper *(Singh, G. and Supriya, 2013)* has a similar approach to this however they include Triple DES as well as DES, AES and RSA. Interestingly they found that AES was more secure than DES as well as 3DES with both *(Paradesi Priyanka et al., 2022, pp.295-300)* and *(Singh, G. and Supriya, 2013)* ranking RSA as the least secure. I will be interested to see how my research will differ. Another approach by (Padmavathi, B. and Ranjitha Kumari, S., 2013) is to rank AES, DES and RSA based purely on time and buffer size. I do not believe this is enough criteria to accurately rank each algorithm against each other. Papers *(Paradesi Priyanka et al., 2022, pp.295-300)* through to *(Padmavathi, B. and Ranjitha Kumari, S., 2013)* only test algorithms on text or binary files, albeit of various sizes.

There is very limited data on audio encryption however I was able to find a paper *(Rizvi, S.A.M, Hussain, S.Z. and Wadhwa, N., 2011, pp.76- 79)* that tests both AES and TwoFish against various factors using text, image and audio files. This is a very in-depth study of these 2 algorithms and the data shows that TwoFish is almost equal to AES in everything except audio encryption where it surpasses it. This is probably due to the fact that AES almost was TwoFish *(Schneier B., 2019)* where it was only beaten by what was previously known as “Rijndael Block Cipher Family” referring to a selection of ciphers which are now know as AES-128, AES-192, AES-256. *(USA Department of Commerce, 2023)*.

Methodology

Approach

To develop my project, I used a modified agile model, using weekly progress updates and designed each individual part of the software slowly, testing as I go, ensuring it works as intended. For requirement gathering, I used a brainstorming method where I write down functional and non-functional requirements and develop on those ideas. As I test my software whilst developing, I will update a mirror document of the brainstorming with progress updates and completion dates.

Technology

I will not require much technology for this project, just access to python for the coding, GitHub for version management and Microsoft Word/Google Docs for documentation.

Version Management Plan

For version management, I will be using a GitHub repository that I will be making regular commits to. This can be found at:
https://github.com/ryanbutbored/ComputingProject

The source code and other documentation can also be found at the Google Drive folder:
https://drive.google.com/drive/folders/17DeizhKS4BJnlb1lybjJ13VMIqkyzF_H?usp=drive_link

Brief Description of Algorithms

Caesar Shift

This is one of the oldest, simple, and most well-known algorithms available. The key (n) is simply a number between 1 and the length of the alphabet used. Then the letters are shifted n times to the right. For example, with the alphabet "abcd" and a key 2, the letter a becomes c and the letter d becomes b.

Substitution Cipher

This is also a very old algorithm and very simple. The key (n) is simply a shuffled version of the alphabet the message is written with. You then take each letter's position in the alphabet and swap it with whatever is in that position in the key. For example, with the alphabet "abcd" and the key "cadb" the letter a becomes c and the letter d becomes b.

Vigenère Cipher

This algorithm uses a special tool called a *tabula recta*, a table with every permutation of caesar shift possible within the current algorithm. Each row and column correlates to a letter of the alphabet. For example, using the table below and the alphabet "abcde", and using the key "bed", the plaintext "bad" would become "ceb" using each letter from both the key and the plaintext to determine the row and column to add to the ciphertext.

	a	**b**	**c**	**d**	**e**
a	a	b	c	d	e
b	b	c	d	e	a
c	c	d	e	a	b
d	d	e	a	b	c
e	e	a	b	c	d

Rail-Fence Cipher

This algorithm is fairly unique in the fact that the key is a number of "rails" to use when encoding/decoding the message. Using a key of 3 and the message "cryptography" the message would become "ctarporpyygh " as you see below. To get this message, you write the message one character at a time, going up and down (and always across) the rails then when you're done read across, rail by rail.

c				t				a			
	r		p		o		r		p		y
		y				g				h	

Enigma

This is actually not an algorithm, but rather a machine that can be treated as one. There are 5 key aspects to this machine that form the key. Plugboard settings, this controls the "swapping of letters" for example a plugboard setting of "h-a" would change an h or a to an a or h respectively when it is first input into the machine and if any letters are encoded into them, then it also swaps them on the way out. Rotor positions control the order in which each rotor is input into the "machine" and therefore effect the way each letter is encoded. Key and Ring settings both act as a caesar shift on each side of the rotor, encoding the letter on the way into the rotor and on the way out. You can find a much more in-depth explanation with examples at (Sale T. (no date)).

RSA

This is the first of the modern algorithms that I recreated. How it works is fairly simple in comparison to the other modern algorithms. The key is 3 separate numbers, the product (n) of 2 1024-bit prime numbers (p and q), a separate number (e) that is coprime with the number (p - 1) * (q - 1), and a final number (d) such that (e * d) MOD (p - 1) * (q - 1) = 1. The public key is the format (e, n) and the private key is (d, n). Encryption and decryption are incredibly simple, with you simply converting the text into an integer. You then put that integer to the power of either e (encryption) or d (decryption) and mod the result by n. The result of that is either your encoded or decoded text, represented as an integer. You can find an excellent example of this using much smaller numbers than you usually would at (*RSA Algorithm Example*, (no date)).

DES

This block-cipher used to be very common among data encryption uses, before being replaced by AES. The key is a 64-bit long string of bits and is expanded into 16 48-bit long keys. This is done by first permuting the key using PC-1, returning a 56-bit key where the first bit is the 57th bit of the original key, the second is the 49th bit, and the last is the 4th bit. Then you split this key into two halves, C0 and D0, then perform a series of left shifts according to a schedule to create new subkeys from 0 to 15. Now for each pair C_nD_n, use another permutation table (PC-2) in the same way as PC-1 to create 16 48-bit keys that will be used to encode the plaintext.

For the encoding process, you split up the plaintext into 64-bit blocks and encode each one separately. You start by permutating the block, similar to the keys, with the table IP. Again similar to the keys, you split it into two halves L_0 and R_0. Then to generate L_{16} and R_{16} you follow the formula $L_n = R_{n-1}$ and $R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$ where K_n is the nth key and f is a function. To perform the function f, you use the e bit-selection table to expand the 32-bit plaintext into 48-bits and XOR the result with K_n. You then split the 48-bit result into 8 6-bit segments which act as coordinates to 8 separate sub-boxes where the first and last bit specify the row, and the rest specifies the column. This outputs a 32-bit result that you permutate with another table (this time just called P) that returns another 32-bit sequence of bytes. This is the final output of the function f. When you have L_{16} and R_{16} you put them back together in reverse as $R_{16}L_{16}$ and apply one final permutation using the table IP^{-1}. The result of this is your ciphertext. To decode this you simply apply the same steps, but by reversing the subkeys you generated at the beginning.

You can find all the tables, key schedules and an example to work through at: https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm.

Triple DES

Once you understand DES, this cipher is incredibly easy to understand. Simply put, it is when you have either 2 or 3 keys and perform a sequence of encryptions or decryptions using DES with the separate keys. For example, with the 3DES in this project, you have two keys (K_0 and K_1). You then encode each block with K_0, decode it with K_1 then encode it again with K_0. To decode you once again just do the same steps in reverse, so decode with K_0, encode with K_1 and decode again with K_0.

Blowfish

Blowfish was originally designed by Bruce Schneier as a substitute/replacement for DES, thus they bear some similarities. The key can be anywhere from 42 bits to 448 bits, and is expanded using an substitution box generated by the digits of pi. Then you split the plaintext into two halves and encrypt the plaintext using 16 rounds and the first 16 of the 18 subkeys.

Finally, you go through a post-processing stage where you reverse the orders of the halves, XOR them with the remaining keys, and then that is your ciphertext. This algorithm is very poorly documented compared to the rest, but you can find a good example with sample solutions at: https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/.

AES

AES consists of multiple rounds, each also consisting of multiple steps, some of which are out of the scope of this report. However, an in-depth explanation of this, and all table/schedules used, is available at the USA Department of Commerce (2023) publication for AES, found at https://www.nist.gov/publications/advanced-encryption-standard-aes-0. For different variations of AES, from 128-bit AES to 256-bit, the number of rounds changes. For AES-128 you would perform 10 rounds, whereas for AES-256 you would perform 14 rounds.

To begin with, you must expand the key using the AES key schedule. This will provide you with enough keys for each round, plus one (11 for AES-128, 15 for AES-256). Then before starting the rounds, you perform the function, AddRoundKey, where you XOR the current plaintext with the key at use in that round (or the first key for use before rounds).

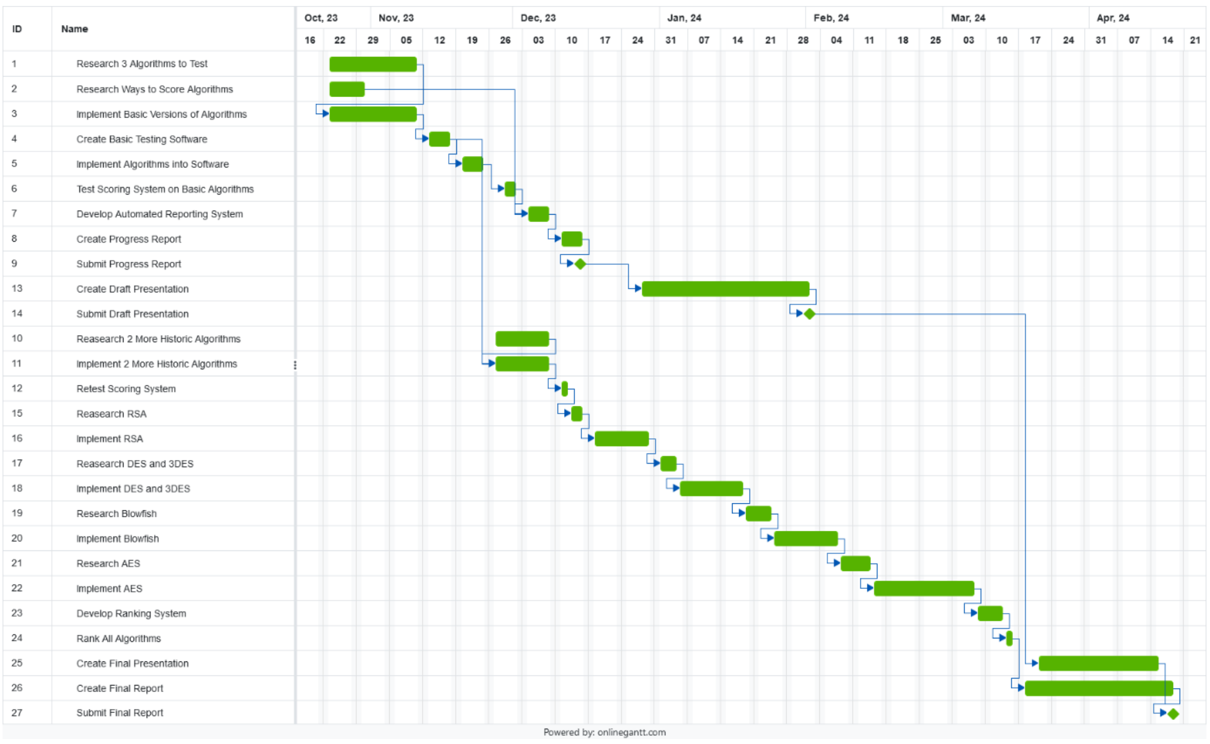
After that, for n rounds, where n is the number of rounds your version of AES uses minus 1, you perform 4 functions in this order: SubBytes (swap data in the matrix representing the plaintext using an 8-bit substitution table), ShiftRows (left shift of the rows in the matrix, where the number of times you perform the left shift is dictated by the row number, i.e. zero left shifts for row 0, and three for row 3), MixColumns (for more information see the publication above), and AddRoundKey. For the final round, you simply perform the same operations in the same order without MixColumns; the result of this is your ciphertext.

Once you understand how to encode with AES, decryption is fairly simple to understand. Each function with the exception of AddRoundKey has an inverse counterpart that simply reverses the effect of the normal function. These are called InvMixColumns, InvShiftRows and InvSubBytes. Once you have these functions, you simply reverse the order of the keys and perform the functions’ counterparts in reverse order to recover the plaintext.

Professional Issues

Project Management

Gantt Chart



Deliverables

My deliverables include:

- Testing Software
- Source Code for 10 algorithms
- Final Report
- Final Presentation

Risk Analysis

Risk	Risk Level	Probability	Solution
Program may crash to incorrect RSA key	Medium	Medium	Implement key validation for RSA in the future
Program data may be inconsistent depending on the type of machine used	High	Low	Scale results to a normal distribution, so that algorithms are compared against a base speed, dependant upon the processing power of the machine used.

Professional Issues

BCS Guideline	Reflection on The Project
Public Interest	As this project is using open-source components, anyone can use this program to either recreate the data in this report or use the algorithms themselves for personal use.
Personal Competence and Integrity	As people this program is open-source, any and all people are free to make modifications, provided those modifications are unbiased and also freely available for other people.
Duty to Relevant Authority	This project follows both GDPR and BCS guidelines, and no personal data is collected at all, so there is no risk at all when it comes to worrying about safety.
Duty to the Profession	As this follows all other BCS guidelines, these are met by default, and once again other developers are free to modify this program to fit their needs.

Results

Security Rating

Due to modern security being incomparable to historical methods, they are ranked separately. Modern security will be ranked on security against a computer, whereas historic methods will be ranked on their ability against a human, as computers can break all of them within seconds.

Modern

1. **AES**
   This is the most secure algorithm on this list, as it is the current standard. It has never been broken and there are currently no known attacks that would work in practice. You can find information on attacks that have been theorised on the Wikipedia page for AES: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Security.
2. **RSA**
   This algorithm is also incredibly secure as it involves factorizing incredibly large numbers that while technically very easily done, it would just take too long. It is said that it would take a regular computer longer than the heat death of the universe to crack just one key, though I have not seen any evidence backing up this statement. There is a worry that quantum computers may be able to crack this encryption, however they are still only theoretical.
3. **Triple DES**
   This algorithm, whilst being very secure, is vulnerable to a variety of attacks (Chowdhury D., *et al.*, 2022) such as sweet-32 attacks and man in the middle attacks. However, these attacks are very space inefficient with man in the middle having a space efficiency of 2^{112}, therefore requiring massive amounts of computing power to break this encoding.
4. **Blowfish**
   This algorithm was very close on my rankings to 3DES as it has a very similar vulnerabilities and equally the vulnerabilities have the same inefficiencies. However, ultimately I believe it is less secure due to its smaller key size which may also leave it vulnerable to birthday attacks as well as other attacks named above.
5. **DES**
   DES is the lowest ranked on this list, because there are numerous attacks that work against it, namely differential cryptoanalysis, linear cryptoanalysis, and Davies' attack. As of 2017, it can be broken by certain (and very expensive) builds, in 25 seconds. The average computer can break this algorithm in anywhere from 2-15 days. You can find more information about DES vulnerabilities here: https://paginas.fe.up.pt/~ei10109/ca/des-vulnerabilities.html

Historical

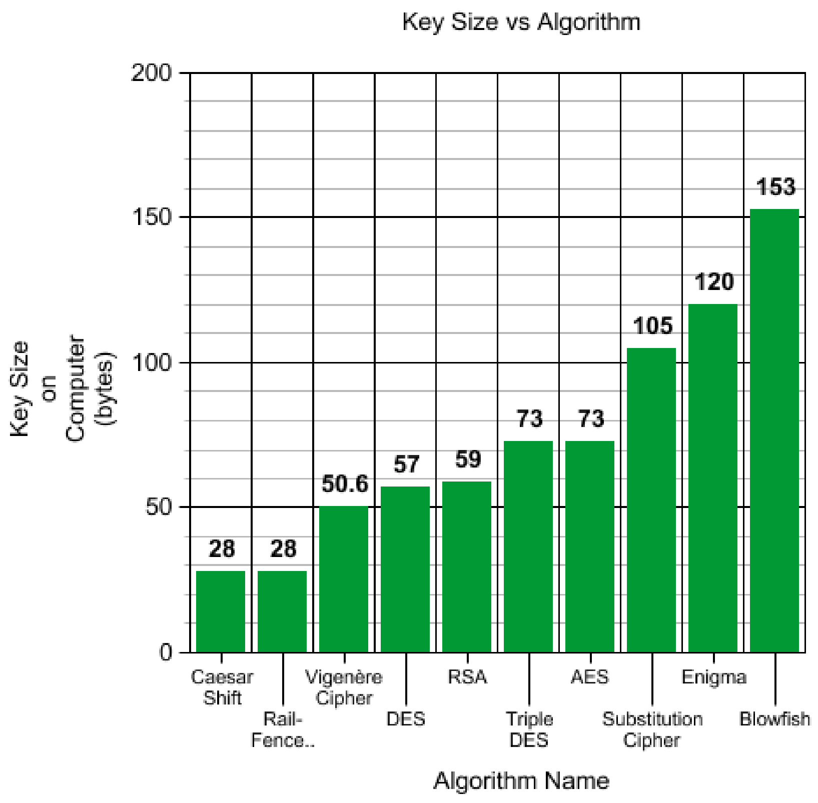
1. **Enigma**
   This is by far the most secure algorithm in this list, with it famously taking the British Military the building of a machine to break it. However, it has been broken by hand before by a group of Polish cryptographers, though this was a less secure version than the version I have reproduced. The main flaw with enigma, though not one that would assist a human when decoding it, is that a letter will never be encoded as itself. You can find more information about the history of enigma on Bletchley Park's official website (https://bletchleypark.org.uk/our-story/enigma).
2. **Vigenère Cipher**

Despite being one of the oldest algorithms on this list, it is rated so highly due to it being a form of a polyalphabetic cipher, which severely reduces the effectiveness of attacks like frequency analysis. Unfortunately, this only applies when the user has used a key of sufficient length, as with the repeating nature of this algorithm, if the key is too short this can create patterns in the ciphertext that can be used alongside frequency analysis to crack this cipher.

3. **Substitution Cipher**
   In this age, calling this algorithm secure is not possible due to its ability to broken trivially with frequency analysis. However, it is still more secure then both Rail-Fence and Caesar Shift as both of these can be brute forced. However, frequency analysis depends strongly on the length of the ciphertext so if a message is short there is less chance of it being broken quickly, however if a message is too short then it can be brute-forced. This does also come with a bonus though, as if the message was just the word “crypto” and an attacker brute forces it, they may come up with a completely separate six letter word, so long as all characters are unique, such as “failed”.
4. **Rail-Fence Cipher**
   This is one of the least secure encoding methods ever invented, though not surprising as it was invented by the Ancient Greeks (see Crypto-IT at http://www.crypto-it.net/eng/simple/rail-fence-cipher.html for more information). The reason it is so insecure is the fact that it simply shuffles the letters, as opposed to switching them with other letters. The key size can also be very easily exploited, as a key too large will not shuffle the letters enough so the message may even be readable without need for decryption, and a small key can be easily brute forced by any human in minutes. The only reason that I believe it to be more secure than a Caesar Shift is that it is not as well known, and therefore attackers may not even know to try to decode it, as unlikely as this is.
5. **Caesar Shift**
   This is the least secure and also probably the most famous encoding method that I have recreated for this project. Its lack of security comes from the lack of keys that are possible with this algorithm, for modern day English there are 26 possible keys. This makes it trivial to brute-force in minutes, or even less if the key is a low number.

Key Size

NOTE: The values in these results are not the actual bit size of the key, rather the size of the keys stored as either a string or an integer using python. It should be noted as well that the RSA value is not accurate as this is the average size of a filename as opposed to the size of the file. Because the average bit size for an RSA key is dependant on the variation used, it could be anywhere from 1024-4069 bits (128-512 bytes).

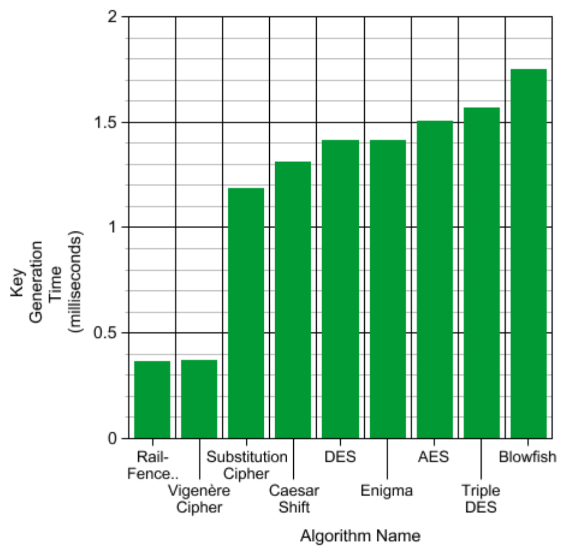


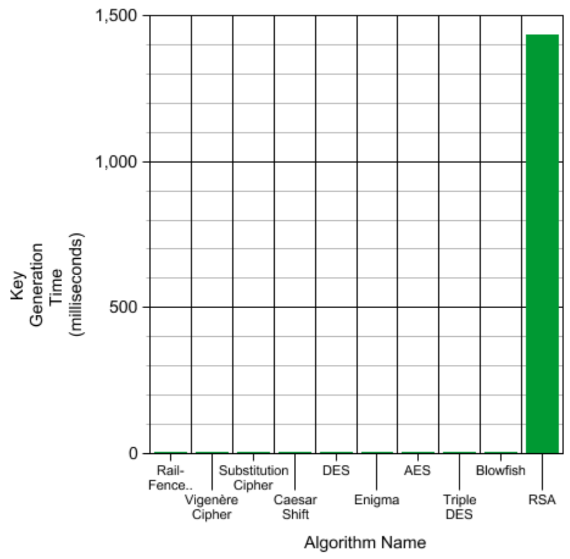
One of the only reasons to worry about key size is when you are dealing with something in the IoT field, where security is paramount, but there is not a lot of memory/processing power to use. In this case you would need a lightweight, small, and fast algorithm. For this purpose, I would recommend DES as it is fast, effective and does not require too much computational power, though it does require more power than what would be considered optimal.

As you can see by the graph above, the key size changes heavily depending on the different algorithms but does not necessarily dictate the security of the algorithm. This can be seen by the substitution cipher having a larger key than AES, DES and Triple DES, all of which are infinitely more secure. It also does not speak to the speed of the algorithm, as a substitution cipher is significantly faster than any of the modern algorithms. Particularly AES which is one of the slowest algorithms, has a key a lot smaller than the substitution cipher.

Key Generation Time

NOTE: As you can see by the graph on the right, the value for RSA dwarfs all other values. This is why for most comparisons; I will be using the graph on the left.



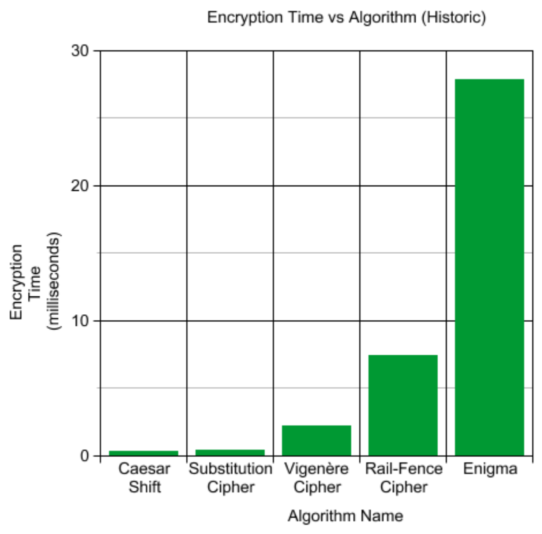


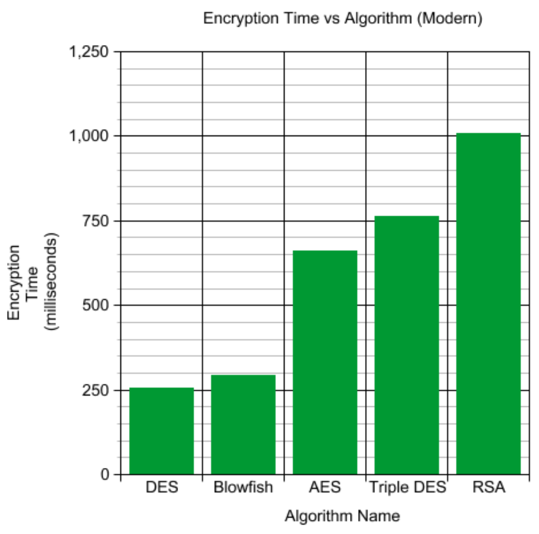
Key Generation Time is important in practical uses, for example when connecting to a server, or another client. If a key generates too slowly then sites will lose users, or potentially have connections time out. However, when connecting to a secure server, you also need to consider how secure it must be. Therefore, based on these results, the best choice would be AES, with it being the most secure and just over 1.5 milliseconds per key.

Interestingly, all algorithms, with the exception of RSA, generate their keys within roughly the same time frame. This means that whilst there are differences, unless they are being applied at mass, the key gen times would not have an effect on the algorithm you choose.

Encryption Time

NOTE: There are 2 graphs for this category, as the modern algorithms are immensely more complicated than the historic algorithms and therefore dwarf them on the graph and make it unreadable.



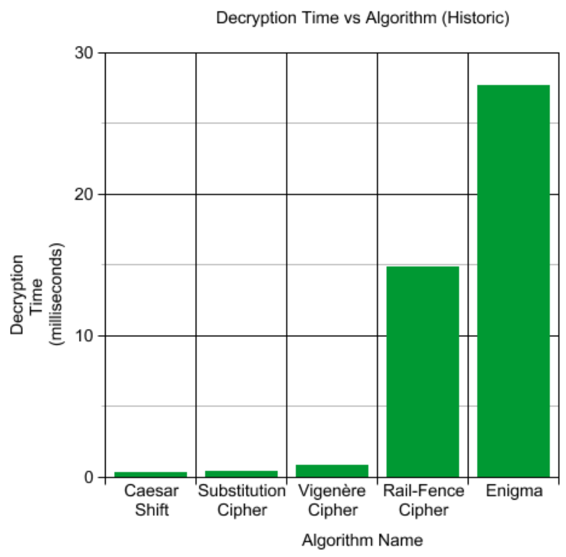


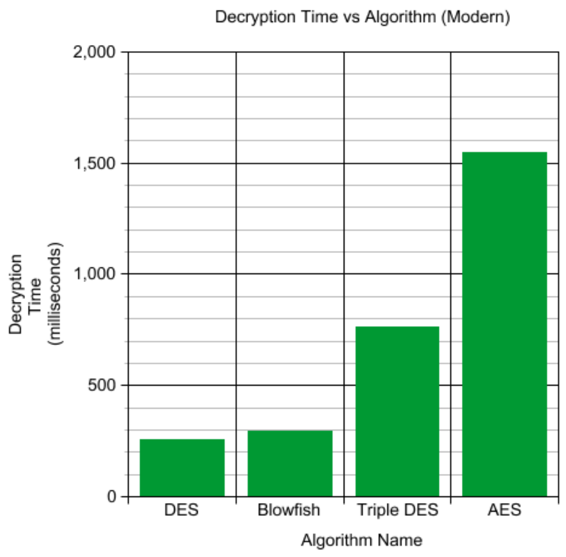
Encryption Time is crucially important in any application of encryption, be it IoT, to a Government Website, to Amazon.com. This is paramount to user experience and with the function of the website itself. Again, I would recommend AES as it is the perfect balance between security and speed.

As these 2 graphs dictate, security and time is a compromise. The more secure and complex the algorithm, the longer it takes. This is further evidenced by the difference between the two graphs, with the longest time on the historic graph (Enigma) taking less than 30 milliseconds and the shortest time on the modern graph (DES) being over 250 milliseconds. Unfortunately, this much of a difference means that unlike most of the other categories, I cannot compare historic with modern on a single graph.

Decryption Time

NOTE: As above, there are 2 graphs for this category, as the modern algorithms again dwarf the historic algorithms. Also, RSA has been omitted, due to it being twenty times longer than AES. This is due to pythons limited ability to read in enormous numbers from the decryption output and perform conversions on them.



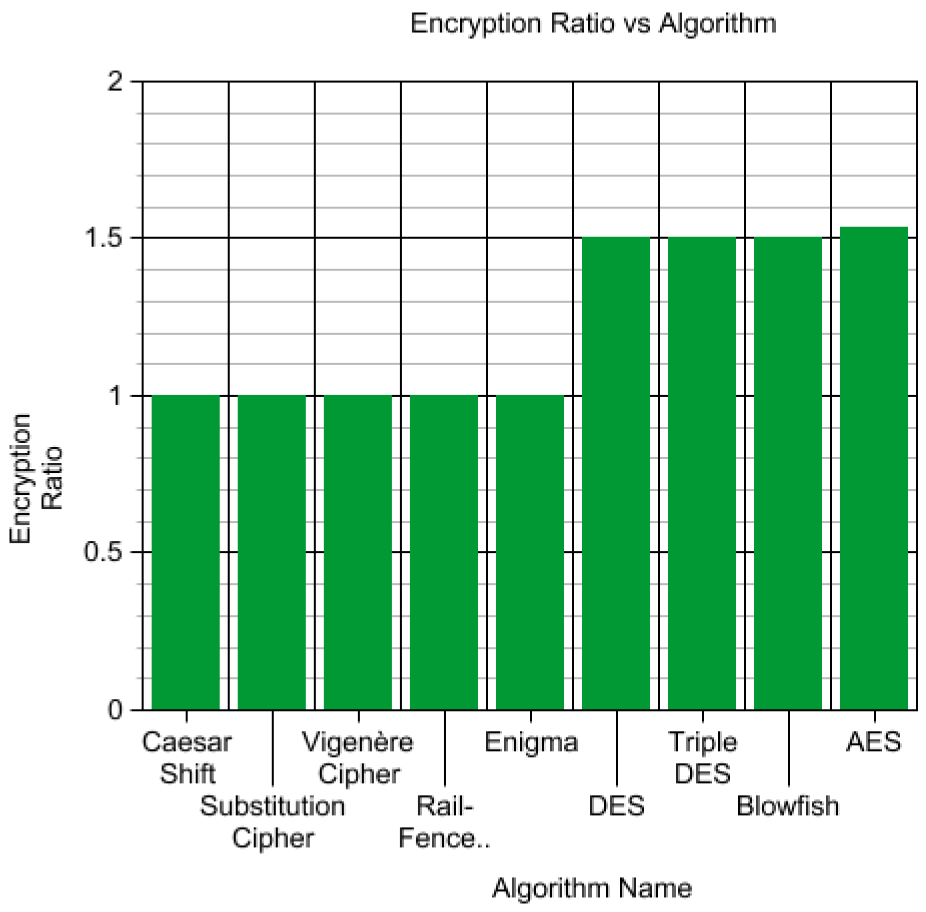


Decryption time is important for all the same reasons as encryption time, after all decryption is just the inverse of encryption. For this reason, again I believe AES is the best choice for any application where security is invaluable and DES if security is less (but still very) important and time is more of a factor (perhaps an informative webpage like Wikipedia where the only personal data needing to be encoded is a user's IP address).

Interestingly, some results differ from what is expected. For example, the decryption of Vigenère if a lot faster than the encryption, despite it just doing exactly the same thing but backwards. Another difference is that Triple DES is faster than AES and is not three times that of DES, despite it being exactly the same but three times. This is most likely because it is only parsing two separate keys instead of three.

Encryption Ratio

NOTE: Once again RSA has been omitted from this graph as due to the size of the numbers in play the encryption ratio of RSA tends to be in the range of 300-400. In the omitted data, the average encryption ratio of RSA was 331.61 (2dp).



Encryption Ratio is very subtlety important in the IoT industry. Where IoT devices typically have less memory and computational power, so if the ciphertext is too much bigger than the plaintext, this could cause issues. This is why, whilst it is not on the graph, RSA would be an incredibly poor choice whereas any of the other modern algorithms would suffice, though I recommend DES, for its security, speed, and small key size.

This is the first graph where you see a definitive difference between modern and historical, with modern encryption ratios being around 1.5, as they encode letters into bytes, and all historical being exactly one, as they all encode one letter into another. This is a really interesting as you can see evidence of the difference between the use cases for the two types of algorithms, one for encoding letters and messages over analogue communication, and the other for computers, encoded into the “languages that computers understand”.

Conclusion

In conclusion, it is clear that for most applications, AES is the clear choice as it has a small key size, allowing the key to be generated almost instantly, the encryption and decryption speeds are both incredibly fast, meaning that data can be encrypted securely without affecting the user experience at all. The encryption ratio is also fairly small, meaning that the computational power required for this algorithm is almost optimal, despite its level of complexity.

However, there are scenarios where I would recommend alternative encoding methods, namely within an IoT setting, or for use within a legacy project. If it were in a IoT environment, I would suggest to use of DES or triple DES, as these both do not require much memory or computational power and are very fast as encrypting data, though 3DES is not as fast as AES due to there being three separate encryptions.

For use in a legacy project, the algorithm you would use entirely depends on the era you are working with and also the type of project you are creating, for example if you were building something dependent upon SSH, a good algorithm to use would be a 512-bit version of RSA, as this could easily be handled by most machines, and SSH is used to having RSA as a standard. However, if the project was recreating an old webpage, then DES would be better suited for this as it is still supported by most (if not all) browsers and this would fit with the theme of the project, as this used to be the standard before it was deemed no longer secure.

Future Work

There is lots of potential for future work within this project. For example, you could include more algorithms, for example if I was to use another modern algorithm, I would want to use TLS (transport layer security) as the results of this would be extremely interesting as this is the standard for IoT and MQTT applications.

It would also be interesting to use these algorithms to encode more than just text. For example, you could encode binary files, images and audio with these algorithms, and test their ability against each other with these tasks. However, the algorithms would need modification beforehand.

Bibliography

1. Panda, M. (2016) “Performance Analysis of Encryption Algorithms for Security” *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, Paralakhemundi, India, pp. 278-284, doi: https://doi.org/10.1109/SCOPES.2016.7955835
2. G. Yadav and A. Majare (2016) “A Comparative Study of Performance Analysis of Various Encryption Algorithms” *International Journal on Recent and Innovation Trends in Computing and Communication*, 5 (3) pp. 70-73 Available at https://ijritcc.org/download/conferences/ICEMTE_2017/Track_2_(EXTC)/1487794878_22-02-2017.pdf
   Accessed: 09/03/2024
3. M. Paradesi Priyanka et al., (2022) "A Comparative Review between Modern Encryption Algorithms viz. DES, AES, and RSA*," 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES),* Greater Noida, India, pp. 295-300, doi: https://doi.org/10.1109/CISES54857.2022.9844393
4. Singh, G. and Supriya, (2013) “A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security” *International Journal of Computer Applications*, 67(19). Available at: https://research.ijcaonline.org/volume67/number19/pxc3887224.pdf
   Accessed: 09/03/2024
5. Padmavathi, B. and Ranjitha Kumari, S. (2013) “A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique” *International Journal of Science and Research*, 2(4).
   Accessed at: https://www.ijsr.net/archive/v2i4/IJSRON120134.pdf
   Accessed: 09/03/2024
6. Rizvi, S.A.M, Hussain, S.Z. and Wadhwa, N. (2011) ‘Performance Analysis of AES and TwoFish Encryption Schemes‘ 2011 International Conference on Communication Systems and Network Technologies, Katra, India, pp. 76-79.
   doi: https://doi.org/10.1109/CSNT.2011.160
7. Schneier B. (2019). *Schneier on Security: Twofish.*
   Available at: https://www.schneier.com/academic/twofish/.
   Accessed: 10/03/2024
8. USA Department of Commerce (2023). *Advanced Encryption Standard (AES).*
   Available at: https://www.nist.gov/publications/advanced-encryption-standard-aes-0.
   Accessed: 10/03/2024
9. *RSA Algorithm Example* (no date).
   Available at: https://www.cs.utexas.edu/users/mitra/honors/soln.html
   Accessed: 04/04/2024
10. Sale T. (no date) *The Principles of the Enigma.*
    Available at: https://www.codesandciphers.org.uk/enigma/enigma1.htm
    Accessed 04/04/2024
11. Choudhury D., *et al.* (2022) ‘DeCrypt: a 3DES inspired optimised cryptographic algorithm’ *Journal of Ambient Intelligence and Humanized Computing* 14(5) pp: 1-11
    doi: https://doi.org/10.1007/s12652-022-04379-7