JOURNAL OF
EMERGING INVESTIGATORS

Article

Comparative analysis of the speeds of AES, ChaCha20, and Blowfish encryption algorithms

Maninder Kaur[1], Nicholas Fishel[1]
[1] Hamilton Southeastern High School, Fishers, Indiana

SUMMARY

With our increasing reliance on digital data and sensitive information, there is a paramount need for robust data protection methods. Our study examined encryption algorithms and their effectiveness in safeguarding data in a fast-paced manner. Our core objective was to compare the speed of three prominent encryption algorithms: Advanced Encryption Standard (AES), ChaCha20, and Blowfish. We hypothesized that the ChaCha20 encryption algorithm would enable the fastest encryption and decryption among the three. To test this, we conducted a set of trials using lorem ipsum text files of various lengths. Each algorithm was subjected to rigorous testing to assess its encryption and decryption times. For each file length text file, each of the encryption algorithms would be tested on the file 10 consecutive times and the average was accessed. While each algorithm demonstrated unique strengths (such as faster encryption time, faster decryption times, similar encryption/decryption times, easier implementation of pushing files to be encrypted into the algorithm, ease of use/coding, etc.), ChaCha20 emerged as a clear winner. These results offer valuable context to decision-makers in sectors in need of implementing security for their data (large technological corporations data, academic institutions' student information, medical facilities' patient charts, etc.) seeking a rapid encryption solution.

INTRODUCTION

In the digital era, it is vital to ensure the security of our online data against unauthorized users. There were 1,802 global large-scale data breaches (breaches that affect a large amount of people, compromise extremely sensitive data, and/or tremendously impact the victims in a negative way) reported in 2022, leading to numerous individuals having their most valuable information exposed (1). One security measure that could be implemented to protect data is encryption algorithms. Defined as methods of converting data from plaintext (unencrypted) to ciphertext (encrypted), these algorithms play a pivotal role in safeguarding sensitive information from malicious intent (2). Encryption algorithms render plaintext data incomprehensible to anyone without the appropriate decryption key. Such encryption algorithms play a crucial role in safeguarding sensitive information across various types of data files, including text documents, images, videos, audio files, and executables.

Yet, each file type has a different way of how the data looks after encryption, even though they all follow the same process. Regardless of the file type and the manner in which the data is depicted, encryption algorithms transform these files into ciphertext, solidifying a wall of security between attackers with intent to harm (3, 4). Encryption techniques create a security barrier against malicious intent by ensuring that even if attackers gain access to encrypted data, they cannot discern its original meaning without the appropriate decryption keys, regardless of the type of data being protected (2, 5).

Three encryption algorithms were selected for this study: Advanced Encryption Standard (AES), ChaCha20, and Blowfish, all of which are symmetric encryption algorithms (4). Symmetric encryption uses the same key to both encrypt and decrypt a file, while asymmetric encryption algorithms utilize different keys for the encryption and decryption processes. We only tested symmetric encryption algorithms because they are generally faster in encryption and decryption (5, 6).

A "bit" is a unit of computational information (7, 8). Computers understand commands through a collection of 1s and 0s. For example, the 4-bit sequence of 1000 would mean the number 8. A key, or what allows encrypted data to be reversed to the original plain text, consists of a sequence of bits. Without the key, an encrypted piece of data cannot be decrypted to its original form. In this study, when we stated that we would be utilizing AES 128-bit, that meant that the key consisted of 128 bits that allowed encrypting/decrypting the chosen file. The larger the number of bits, normally the stronger and more complex the encryption (8). The key sizes provide possible keys. For example, an AES 128-bit key size can produce 2^{128} possible keys.

Despite all being symmetric encryption algorithms, AES, Blowfish, and ChaCha20 have distinct characteristics that influence their performance within different scenarios such as its use in wireless network security or secure browsing. Hardware support is a key differentiator; AES-128's extensive hardware support provides a performance advantage in many scenarios, such as its use in wireless network security or secure browsing, while ChaCha20's software efficiency makes it preferable in contexts where hardware acceleration is unavailable (4-6). Although Blowfish can be implemented in hardware, it does not have the same level of widespread support as AES, which may limit its performance in certain applications. Block size also plays a significant role in security and performance; Blowfish has a smaller block size of 64 bits, which is generally considered less secure for large-scale data encryption due to the increased likelihood of repeating patterns that can be exploited in attacks (5). In contrast, larger block sizes, like AES's and ChaCha20's 128 bits, mitigate

https://doi.org/10.59720/23-246

these risks (5, 8, 9). Hardware support and acceleration enhance performance by offloading cryptographic operations to specialized hardware, which is faster and more efficient than software-based methods (8, 9). Understanding these differences helps in selecting the most appropriate encryption algorithm for a given use case.

Blowfish is a symmetric-key block cipher encryption algorithm designed by Bruce Schneier in 1993 that uses variable key sizes, with the recommended key size being 128 bits (4). Blowfish operates on 64-bit blocks of plaintext and supports variable key lengths, making it adaptable to different security requirements. One of the key strengths of Blowfish is its simplicity, which allows for a relatively easy implementation and fast encryption and decryption processes (10). Additionally, Blowfish has undergone extensive cryptanalysis and has proven to be resilient against various attacks, further enhancing its reputation as a reliable encryption algorithm (11,12). While newer algorithms like AES and ChaCha20 gained popularity in recent years, Blowfish remains a noteworthy historical milestone in the field of cryptography and continues to be used in various applications where speed and security are essential (11).

Another one of the encryption algorithms we tested was AES, a symmetric-key encryption algorithm with key sizes of 128, 192, or 256 bits (4, 13). AES employs block cipher encryption, dividing plaintext into fixed-size blocks and applying multiple rounds of transformation to produce ciphertext (13). It was selected by the U.S. National Institute of Standards and Technology (NIST) in 2001 as a federal government standard due to its robustness, speed, and suitability for a wide range of applications. AES was developed as a replacement for the aging Data Encryption Standard (DES), which had become insufficient for modern security needs. The selection process for AES was part of NIST's efforts to ensure a secure and efficient encryption standard for the future. Its versatile security levels can be showcased with its multiple key sizes (128, 192, and 256), providing different levels of security based on the different situations. The larger the key size becomes, the longer the encryption and decryption processes may take (4, 14). This is another reason why AES 128 was chosen for testing in this manuscript alongside the other algorithms (4, 14). AES can achieve encryption speeds of several gigabytes per second, hitting standard operational benchmark goals for modern encryption algorithms (4, 13, 14). With these characteristics in mind, the real-world applications range from basic TLS (Transport Layer Security)/SSL (Secure Sockets Layer) cryptographic internet security protocols for secure internet browsing, to storage protection, to secure messaging on social media (15-17).

ChaCha20 is a symmetric encryption algorithm designed to provide high security and fast performance with support for a 128-bit or 256-bit key size. ChaCha20 is also a stream cipher, meaning it encrypts files by pushing every binary digit in the data through the algorithm via a cryptographic key (18). This characteristic is fundamentally different from two block ciphers introduced earlier (Blowfish and AES): block ciphers puts data into blocks and encrypt them as groups (19). ChaCha20 was created by Daniel J. Bernstein in 2008 as part of the eSTREAM project (2004-2008), an EU-funded initiative led by ECRYPT, aiming to create new stream ciphers as replacements for older algorithms. The project had a significant impact on cryptographic research and created multiple notable ciphers, one being ChaCha20 (a variation of another cipher created at the time, Salsa20) (18). ChaCha20 is known for its simplicity and ease of implementation while still offering strong security against various cryptographic attacks (4, 18). The design of ChaCha20 aimed to achieve a balance between security, speed, and resistance to side-channel attacks (attacks that exploit physical information leaked during the execution of cryptographic algorithms rather than attacking the algorithms themselves) (19). ChaCha20 is widely used in various applications, including secure communication protocols, disk encryption, and digital signature schemes (20, 21).

We chose the three encryption algorithms described here for the following shared characteristics: they are all symmetric encryption algorithms, are widely used, and have different key sizes from which to choose. A previous study compared multiple encryption algorithms, including AES and Blowfish, and assessed the algorithms' performance against known cryptographic attacks in terms of multiple factors, namely architecture, flexibility, reliability, security, and limitation (22). This study found that AES is relatively faster than Blowfish, and that Blowfish may have limitations when the key size is smaller, as a shorter key length can reduce its security against brute-force attacks and make it more vulnerable to cryptographic attacks (22). One of Blowfish's advantages is its variable key length, which ranges from 32 bits to 448 bits, allowing for greater flexibility in security levels. Blowfish is also known for its speed and efficiency, particularly in software implementations, where it can outperform AES in terms of processing speed due to its simpler structure and smaller block size of 64 bits which leads to faster process times in applications that require high throughput (10-12). These prior findings shed light on the strengths and weaknesses of each algorithm, contributing to the understanding of their real-world applicability.

Similarly, another study analyzed the ChaCha20 cipher and its impact on the development of modern cryptographic techniques (23). Given the newness of ChaCha20, this paper was used in our paper to build knowledge about this algorithm. This study examined the mathematical foundations of the encryption algorithm, revealing key findings relevant to its strengths in asymmetric encryption (23). The analysis highlighted that the algorithm's security is significantly influenced by key sizes, with larger keys providing enhanced protection against brute-force attacks. Additionally, the research found that the time required for encryption and decryption varies notably with key size, indicating that while larger keys offer better security, they may also lead to increased processing times. These findings underscore the importance of balancing security and performance when selecting key sizes for effective encryption.

Still, none of these previous studies focused and compared the speed of these three algorithms specifically (22, 23). Our primary objective was to compare three widely adopted encryption and decryption algorithms: AES, ChaCha20, and Blowfish. We hypothesized that the ChaCha20 algorithm would have a faster encryption and decryption time compared to AES and Blowfish. In the real world, files of different lengths are encrypted; thus we tested different file lengths. By running text files of varying lengths multiple times through all three algorithms and recording the average of each trial, we were

https://doi.org/10.59720/23-246

able to identify which algorithm was fastest for encryption and decryption. We found that ChaCha20 did indeed work faster than AES and Blowfish. Unlike the other two algorithms, which differed in their encryption and decryption times, ChaCha20 achieved more balanced times. Additionally, there was no discernible difference in required time as text file sizes increased. Our paper provides a reference to individuals to find which encryption algorithm better suiting their situation relating to text file encryption.

RESULTS

We compared the performance of three encryption algorithms, AES, ChaCha20, and Blowfish, across various text file lengths: sentence, paragraph, page, chapter, section, and book. We tested the encryption and decryption speed of each algorithm through ten trials for each text file length. The average encryption and decryption times, measured in seconds, were derived from the time taken for each encryption and decryption process for each text file length over 10 trials (**Table 1**).

AES consistently exhibited a shorter average encryption time (0.07937 seconds) than Blowfish, but longer than ChaCha20 across all text file lengths. However, its average decryption time (0.00063 seconds) was generally shorter than ChaCha20 and Blowfish.

Blowfish generally performed comparably to AES in terms of encryption time but consistently had longer decryption times. It was notably slower than ChaCha20 for both encryption and decryption.

ChaCha20 consistently outperformed AES and Blowfish in terms of average speed (encryption and decryption time added together) for all text file lengths. When averaging both operations, ChaCha20 had the shortest average time (0.00372 seconds) compared to AES (0.0973 seconds) and Blowfish (0.00892 seconds) (**Table 1**). ChaCha20's encryption and decryption times were relatively balanced and showed consistent performance across different text lengths. Despite the varying sizes of the text files, there was no discernible correlation between encryption or decryption times and file length (**Table 1**). All three algorithms consistently completed their processes in approximately the same amount of time for each of the data files.

DISCUSSION

In this study, we compared the speed of three encryption algorithms, AES, ChaCha20, and Blowfish, when applied to text files of various lengths. Our findings revealed that ChaCha20 was the fastest encryption algorithm, showcasing consistently faster average times compared to AES and Blowfish. While AES demonstrated competitive decryption times to ChaCha20, it generally lagged behind in encryption speed, whereas Blowfish proved slower overall for both processes. These results offer valuable insights for researchers and developers seeking optimal encryption solutions. It was clear that ChaCha20 was the winner, with its combined average encryption and decryption times beating both Blowfish and AES.

There are multiple unique features of ChaCha20 that may have allowed the algorithm to provide the fastest performance out of the three algorithms tested. Stream ciphers are generally simpler, which may make them more efficient at performing quick cryptographic transformations (4, 10). The algorithm employs various Boolean operations, specifically utilizing addition, rotation, and XOR operations—which are different types of computational arithmetic used by computers to perform calculations and transformations (18-21). These operations are computationally parallelized to work quicker than the other two algorithms, which use more complex operations (18, 21). Furthermore, both AES and Blowfish need a fast device capable of hardware acceleration (12, 13). On the other hand, ChaCha20 works regardless of hardware capabilities (23). We performed our experiment on a MacBook, but we note that the results may differ if repeated on an older PC or a modern gaming PC.

Another remaining question concerns why encryption and decryption times for algorithms either differ or remain consistent. AES's encryption and decryption times are notably unequal (**Table 1**). This imbalance is likely due to the differences in the key schedule process for encryption and decryption. While the key expansion for encryption is straightforward, it involves multiple rounds of transformations that contribute to the overall processing time. In contrast, decryption utilizes an inverse key schedule, which, although it may seem more complex, can be executed more quickly because it does not require the same level of additional transformations as encryption (24). As a result, decryption for both AES and Blowfish was faster than encryption. ChaCha20 also exhibited asymmetry in encryption and decryption times, but this asymmetry was less pronounced than in the other two algorithms. This reduced asymmetry is likely related to ChaCha20's design, which employs a simpler and more efficient key schedule that does not require the same level of transformation complexity as AES and Blowfish (18, 21). According to research, ChaCha20's streamlined operations allow for more consistent performance across both encryption and decryption processes, resulting in less variation in processing times (18).

Blowfish was the slowest of the three algorithms, likely due to the mechanics of older algorithms. The key schedule for Blowfish is notably complex, meaning that the process of generating and organizing keys is not straightforward (4, 11). It involves multiple transformations, iterations, and

Length	AES Encryption	AES Decryption	ChaCha20 Encryption	ChaCha20 Decryption	Blowfish Encryption	Blowfish Decryption
Sentence	0.01024	0.0006	0.0028	0.0009	0.097	0.001
Paragraph	0.099	0.0007	0.0028	0.0011	0.098	0.0013
Page	0.099	0.0007	0.0026	0.0009	0.09	0.0012
Chapter	0.09	0.0008	0.0029	0.0011	0.083	0.0017
Section	0.092	0.0008	0.0027	0.0011	0.097	0.0011
Book	0.086	0.0002	0.0027	0.0007	0.086	0.0011

Table 1: Average encryption/decryption time for each algorithm. Data was collected by recording the time taken for each encryption and decryption process for a text file containing a certain length of Lorem Ipsum text. This was repeated for 10 trials, then the data was averaged.

https://doi.org/10.59720/23-246

bitwise operations, making it harder to reverse or predict the subkeys from the original key. In contrast, newer algorithms often simplify operations by replacing entire processes with single operation symbols. Since Blowfish does not adopt this approach, it takes longer to encrypt and decrypt due to its inherent complexity (11). Another remaining question concerns why encryption and decryption times for algorithms either differ or remain consistent. AES's encryption and decryption times are notably unequal (**Table 1**). This imbalance is likely due to the differences in the key schedule process for encryption and decryption. While the key expansion for encryption is straightforward, it involves multiple rounds of transformations that contribute to the overall processing time. In contrast, decryption utilizes an inverse key schedule, which, although it may seem more complex, can be executed more quickly because it does not require the same level of additional transformations as encryption (24). As a result, decryption for both AES and Blowfish was faster than encryption. ChaCha20 also exhibited asymmetry in encryption and decryption times, but this asymmetry was less pronounced than in the other two algorithms. This reduced asymmetry is likely related to ChaCha20's design, which employs a simpler and more efficient key schedule that does not require the same level of transformation complexity as AES and Blowfish (18, 21). According to research, ChaCha20's streamlined operations allow for more consistent performance across both encryption and decryption processes, resulting in less variation in processing times (18).

Surprisingly, we found that the time for encryption and decryption did not change based on file size. Initially, we hypothesized that larger files would take longer to process, expecting a direct correlation between file size and processing time. This unexpected result challenges our initial assumptions and highlights the need for further investigation into the factors influencing encryption and decryption times. One possible explanation for the lack of correlation between file size and processing time is that the hardware acceleration of the device may have been capable of performing cryptographic operations in real time (3, 4). This means that the device can efficiently handle multiple operations simultaneously, allowing it to process data quickly regardless of the file size. As a result, the time taken for encryption and decryption may remain consistent across different file sizes because the hardware can manage the increased data load without a significant increase in processing time (3, 8). Where devices have hardware acceleration and, consequently, can perform cryptographic operations in real time, the relationship between file size and processing time is no longer directly related (3, 4, 8). The ability to process data in fixed-size blocks, leverage parallel processing, and maintain high throughput means that larger files can be encrypted or decrypted quickly, often resulting in processing times that do not correlate directly with the total file size (3, 8). This is particularly important in applications requiring fast and secure data handling (2, 3, 25). This bridges into the second reason, stream and block ciphers are able to split data, regardless of size, into fixed number sets that are transformed (4). Block ciphers create a certain number of blocks regardless of the original contents of the file (10). Stream ciphers, by contrast, encrypt bit by bit. While the latter process is considered less efficient, stream ciphers often use hardware capabilities to encrypt bits faster (4, 10). Hence, there was no clear increase in encryption or decryption time as file size grew.

It has been noted that in many papers similar to ours, the duration of the encryption and decryption processes is often measured and reported in seconds (22, 23). This is an ideal approach rather than choosing time per character, which has some faults. For one, every algorithm initializes prior to encrypting (4). The initialization time for encryption processes can vary based on the file size, potentially skewing results for shorter files. Specifically, shorter files may have a shorter initialization time, which could lead to an overall reduced processing time that does not accurately reflect the actual encryption time per character (4). However, in our experiments, we found no correlation between processing time and the number of characters in the files. This suggests that while initialization time may differ, it does not significantly impact the overall processing time for encryption, indicating that other factors may be at play. Thus, it may not be optimal to compare time per character when evaluating the algorithm's performance, especially if the goal of the algorithm is to achieve consistent encryption and decryption times regardless of the input size. If the algorithm is designed to process data uniformly, focusing on time per character could be misleading, as it may not accurately reflect the algorithm's efficiency in handling varying file sizes or complexities. Naturally, the effect of initialization time on overall processing time becomes more pronounced in larger files. However, small file trials may well show variability due to different overheads, while increasing file sizes make the general performance more consistent across board (3, 4).

Each encryption algorithm has potential limitations. ChaCha20 is known for its efficiency in software implementations, especially on devices without hardware support for AES instructions (4, 5, 10). In some cases, ChaCha20 with a 256-bit key can provide comparable or better performance than AES128, making it an attractive choice for applications where computational efficiency is critical (4, 11, 16, 17). However, due to ChaCha20 being a relatively new algorithm, it might not have undergone the same level of scrutiny as AES (14). This scrutiny is from academic researchers that create and review the ciphers, government agencies like NIST, and the cryptographic industry (4, 13, 22). They evaluate their scrutiny based on strength, mathematical foundations, vulnerabilities, performance, and many more all of which is helpful to create ciphers that can be used and will keep data safe. The weaknesses of AES include increased susceptibility to timing attacks, as well as potentially slower performance in certain applications that lack hardware acceleration and in software-only situations (4, 10, 11). Blowfish is known for its slower performance compared to modern algorithms, and the absence of widespread adoption in critical applications limit its use in scenarios where speed is a priority (4, 19, 23). However, it was included in this comparison due to its historical significance in the development of symmetric encryption algorithms and its continued relevance in certain contexts, such as legacy systems and resource-constrained environments (4, 12).

Limitations of this experiment include the use of pseudocode instead of actual code execution, which might not account for all implementation details and optimizations in specific programming languages and libraries. While the code and pseudocode were developed using an encryption library for simulation purposes, actual implementations can

https://doi.org/10.59720/23-246

exhibit subtle differences that may affect performance (26-31). For instance, different programming languages may have varying levels of efficiency in handling memory management, which can significantly impact the performance of encryption algorithms (32). If these changes are not reflected in the pseudocode, the experimental results may underestimate the actual performance of the algorithms when implemented in a real-world scenario. We expect that real-world encryption algorithms would be much more complex. If one were to test actual encryption algorithms that are available on their everyday computers, the results may vary. Additionally, the experiment focused solely on speed and did not assess the algorithms' resistance to advanced attacks. Real-world implementations also consider other factors, such as memory usage and platform support, which we did not consider in this study. Alongside this, we conducted the experiments on a single machine, and the results may vary on different hardware configurations. A computer with different hardware acceleration, RAM, GPU, or other parameters may produce different results (3, 6, 9). Furthermore, we studied only text that used the Latin alphabet, which may not be completely representative of all languages.

Our research could have other aspects taken into consideration that were not highlighted in this manuscript. For example, future research could evaluate the algorithms' security strength against malware or viruses, including resistance to more advanced attacks. Such advanced attacks include side-channel attacks or attacks include the ability of power consumption of cryptographic operations to reveal information about the key or the text (24). Asymmetric systems also exceed symmetric cryptographic systems in certain contexts such as digital signatures or secure email communication (6). All experiments in this paper were done using symmetric encryption algorithms. While symmetric systems are generally faster, asymmetric algorithms are generally more secure (4). Thus, future studies of asymmetric systems would provide more context in this field. Another consideration is investigating the algorithms' performance in resource-constrained environments, such as Internet of Things (IoT) devices or embedded systems. This could provide valuable insights for practical applications like smart home devices, wearable technology, and automotive systems (25). Additionally, future research could examine other algorithms beyond the three we tested. While this study focused on symmetric algorithms, future research could explore the performance of asymmetric algorithms such as RSA, ECC, and DSA. These algorithms have different key sizes and operational characteristics that could provide valuable insights into their efficiency and security in various contexts (4, 5). For instance, RSA and ECC utilize larger key sizes, which may impact encryption and decryption times differently compared to the symmetric algorithms tested in this study (4). Investigating these algorithms could enhance our understanding of their performance trade-offs and inform best practices for their implementation in real-world applications. Future experiments may utilize these algorithms for testing. Future work could also explore hybrid encryption approaches to combine the strengths of different algorithms. It would also be important to include real-world implementations in various programming languages and platforms, such as Python, Java, C++, and embedded systems like Arduino or Raspberry Pi (32, 33). Lastly, future research can consider different types of data files. A video file is far more complex than a text file, for example, so there is a possibility that a certain algorithm may work better for such file types.

We examined encryption algorithms due to their critical role in safeguarding digital data and were particularly interested in the role of speed in algorithm selection. We identified ChaCha20 as a standout performer in terms of speed for text files of various sizes. Moving forward, continued exploration of encryption techniques and their real-world applications will likely drive advancements in cybersecurity, ensuring a safer digital landscape for individuals and organizations alike.

MATERIALS AND METHODS

Experiments were conducted on a computer with an Intel Core i5 processor, 8 GB RAM, and a solid-state drive (SSD). The operating system used was a macOS Monterey Version 12.6. The experiments were performed using Python 3.x with the 'cryptography' library (version 41.0.1) (26). All Python libraries were updated and run on the latest versions (all compatible with Python 3.11). Code and pseudocode were developed for each encryption algorithm to simulate the encryption and decryption processes (31). The pseudocode for AES, ChaCha20, and Blowfish was adapted from standard Python implementations using the `cryptography` library. Each algorithm was tested ten times on each of the text files, with the size of the file ranging from a few bytes to a few kilobytes.

The experiment used five text files with different lengths of Lorem Ipsum content as the input data for encryption and decryption. The content was created using a Lorem Ipsum generator, which can accurately mimic real-life data (34). Text files were generated of varying lengths to emulate a sentence, a paragraph, a page, a chapter, a section, and the length of a complete book. The text files were preprocessed to ensure consistent formatting and content. The lengths of each file, in the approximate total number of characters, were: 15-20 for sentence, 150-200 for paragraph, 300-700 for page, 3000-5000 for chapter, 10000-20000 for section, and 100000 for book.

We used the following key sizes to ensure equivalency between all algorithms: AES, 128 bits; ChaCha20, 256 bits; and Blowfish, 128 bits. We chose a larger bit size for ChaCha20 due in part to differences in the design and structure of the algorithms. ChaCha20 256 was chosen to be comparable to AES and Blowfish in terms of performance ability (4).

Received: September 29, 2023
Accepted: July 13, 2024
Published: October 8, 2025

REFERENCES

1. "Identity Theft Resource Center's 2022 Annual Data Breach Report Reveals Near-Record Number of Compromises." ITRC, Identity Theft Resource Center, https://www.idtheftcenter.org/post/2022-annual-data-breach-report-reveals-near-record-number-compromises/. Accessed 25 Jan. 2023.
2. "What Is Encryption?" IBM, https://www.ibm.com/topics/encryption. Accessed 25 Jan. 2023.
3. Sruthi, S., et al. "Encryption and Decryption of Text File and Audio Using LabVIEW." *2017 International*

https://doi.org/10.59720/23-246

Conference on Networks & Advances in Computational Technologies (NetACT), IEEE, 20-22 July 2017, https://doi.org/10.1109/netact.2017.8076816.
4. Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography. 2nd ed., CRC Press, 1996.
5. "Encryption Choices: RSA vs. AES Explained." 15 June 2021, https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes. Accessed 25 Jan. 2023.
6. Kumar, A., et al. "Performance Analysis of Encryption Algorithms for Security." *Proceedings of the IEEE Conference,* 1 Oct. 2016, IEEE, https://ieeexplore.ieee.org/document/7955835.
7. "Bit (Communications)." Encyclopaedia Britannica, https://www.britannica.com/technology/bit-communications. Accessed 25 Jan. 2023.
8. "All About Cryptographic Bit Lengths." TCC - Technical Communications Corporation: Network Encryption, Secure Communications, https://www.tccsecure.com/NewsResources/CipherONEBlog/TabId/1222/ArtMID/1578/ArticleID/2/All-About-Cryptographic-Bit-Lengths.aspx. Accessed 29 May 2024.
9. Al Tamimi, Abdel-Karim. Performance Analysis of Data Encryption Algorithms. https://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf. Accessed 25 Jan. 2023.
10. "Difference Between Block Cipher and Stream Cipher." GeeksforGeeks, 20 May 2019, https://www.geeksforgeeks.org/difference-between-block-cipher-and-stream-cipher/. Accessed 25 Jan. 2023.
11. "Blowfish Algorithm with Examples." GeeksforGeeks, 14 Oct. 2019, https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/. Accessed 25 Jan. 2023.
12. Awati, Rahul. "Blowfish." Security, 19 Jan. 2022, https://www.techtarget.com/searchsecurity/definition/Blowfish. Accessed 25 Jan. 2023.
13. "Advanced Encryption Standard (AES)." NIST, https://www.nist.gov/publications/advanced-encryption-standard-aes. Accessed 29 May 2024.
14. Daemen, Joan, and Vincent Rijmen. "The Design of Rijndael." *Information Security and Cryptography,* 2002, https://doi.org/10.1007/978-3-662-04722-4.
15. "SSL and TLS: Designing and Building Secure Systems." EBIN.PUB, https://ebin.pub/ssl-and-tls-designing-and-building-secure-systems-0201615983-9780201615982.html. Accessed 25 Jan. 2023.
16. Paolomatarazzo. "BitLocker Overview - Windows Security." Microsoft Learn, 6 Nov. 2023, https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker. Accessed 29 May 2024.
17. "Apple Platform Security." Apple Support, https://support.apple.com/guide/security/welcome/web. Accessed 25 Jan. 2023.
18. Bernstein, Daniel J. "ChaCha20, a Variant of Salsa20." University of Chicago, Department of Mathematics, Statistics, and Computer Science, https://cr.yp.to/chacha/chacha-20080120.pdf. Accessed 25 Jan. 2023.
19. "What Is a Side-Channel Attack? How It Works." GeeksforGeeks, 31 Mar. 2024, https://www.geeksforgeeks.org/what-is-a-side-channel. Accessed 29 May 2024.
20. Computerphile. "ChaCha Cipher - Computerphile." YouTube, 19 Feb. 2021, https://www.youtube.com/watch?v=UeIpq-C-GSA. Accessed 25 Jan. 2023.
21. Nir, Yoav, and Adam Langley. "RFC 7539: ChaCha20 and Poly1305 for IETF Protocols." IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc7539. Accessed 29 May 2024.
22. Kumari, Shailja, and Jyoti Chawla. "Comparative Analysis on Different Parameters of Encryption Algorithms for Information Security." Proceedings of the Conference, 2015, https://api.semanticscholar.org/CorpusID:212544681.
23. Shi, et al. "Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha." *ICISC'12: Proceedings of the 15th International Conference on Information Security and Cryptology,* 2013, pp. 337–51. https://doi.org/10.1007/978-3-642-37682-5_24.
24. "Cybersecurity of Quantum Computing: A New Frontier." SEI Blog, https://insights.sei.cmu.edu/blog/cybersecurity-of-quantum-computing-a-new-frontier/. Accessed 29 May 2024.
25. Kamal, R. "Securing the IoT Data Landscape: IoT Encryption Algorithms." Intuz, https://www.intuz.com/blog/securing-the-iot-data-landscape-iot-encryption-algorithms. Accessed 25 Jan. 2023.
26. "Cryptography." Cryptography Documentation, https://cryptography.io/en/latest/. Accessed 25 Jan. 2023.
27. "os module documentation." Python Documentation, https://docs.python.org/3/library/os.html. Accessed 25 Jan. 2023.
28. "time module documentation." Python Documentation, https://docs.python.org/3/library/time.html. Accessed 25 Jan. 2023.
29. "Cryptographic Hashes." Cryptography Documentation, https://cryptography.io/en/latest/hazmat/primitives/cryptographic-hashes/. Accessed 25 Jan. 2023.
30. Kaur, Maninder. "GitHub - Themaninderkaur/Research-paper-sources: Code, Pseudocode and Text Files Used in Research Paper." GitHub, https://github.com/themaninderkaur/research-paper-sources. Accessed 25 Jan. 2023.
31. "Rocket Software Documentation." https://docs.rocketsoftware.com/bundle/unidataunibasiccommands_rg_824/page/vmn1685024690247.html. Accessed 25 Jan. 2023.
32. Team, Codecademy. "7 Best Programming Languages for Cryptography." Codecademy Blog, 9 Apr. 2024, https://www.codecademy.com/resources/blog/programming-languages-for-cryptography. Accessed 29 May 2024.
33. Daniel, Brett. "What Are Embedded Systems?" Trenton Systems, 18 Mar. 2024, https://www.trentonsystems.com/en-us/resource-hub/blog/what-are-embedded-systems. Accessed 29 May 2024.
34. "Lorem Ipsum – Generator, Origins and Meaning." Lorem Ipsum, https://loremipsum.io. Accessed 25 Jan. 2023.