Secure Communication Via Cryptographic Systems

Ziyun Ma
zma@nd.edu
University of Notre Dame

Rana Hussain
rhussain@nd.edu
University of Notre Dame

Haotian Wang
hwang35@nd.edu
University of Notre Dame

Manuela Roca
mrocazap@nd.edu
University of Notre Dame

Abstract

In an age where data is the most valuable asset, cryptographic security plays a foundational role in protecting personal, medical, corporate, governmental, and other similar important information from malicious threats. This project explores the design, implementation, and evaluation of four encryption algorithms: Vigenère Cipher, Triple DES (3DES), Advanced Encryption Standard (AES), and Rivest–Shamir–Adleman (RSA) through a hands-on programming approach. Motivated by real-world breaches caused by poor cryptographic practices, our group developed a multi-functional encryption tool with a custom GUI built using pygame GUI, allowing users to input text, binary data, and files for encryption or decryption using various key management options. The Vigenère cipher decryption demonstrates how classical ciphers can be broken with frequency analysis, providing historical context for the necessity of modern encryption. RSA and AES implementations include support for handling file and image data, highlighting the challenges of byte alignment and secure padding. The 3DES implementation illustrates the evolution of symmetric encryption, showcasing a transition from the older DES standard to a more robust, albeit slower, approach. This helps to contextualize the need for algorithms like AES, which offer both security and efficiency. Evaluation metrics include ciphertext overhead and encoding/decoding time across data types. This work not only demonstrates the practical aspects of secure system design, but also reflects on usability, cryptographic strength, and the trade-offs between legacy and modern techniques in real-world applications.

ACM Reference Format:
Ziyun Ma, Rana Hussain, Haotian Wang, and Manuela Roca. 2024. Secure Communication Via Cryptographic Systems. In *Proceedings of CSE 40567/60567: Computer Security*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Motivation: Why Cryptography Matters

Cryptography is a cornerstone of modern digital security. As our personal, financial, and professional lives increasingly move online, the protection of sensitive information has never been more critical. Cryptographic systems safeguard data integrity, confidentiality, and authenticity. These protections help prevent unauthorized access, fraud, and identity theft. When encryption is poorly implemented or not used at all, the consequences can be severe for both individuals and organizations.

Several real-world cases illustrate the importance of strong cryptographic practices. In 2017, Equifax [11], one of the largest credit reporting agencies in the United States, experienced a massive data breach. Attackers exploited a known software vulnerability and gained access to personal information belonging to 147 million Americans. This included Social Security Numbers and credit card details. Much of the data had not been properly encrypted while stored or had been stored completely unencrypted. Therefore the lack of proper encryption-at-rets practices resulted in over 700 million dollars in damages, a major loss of public trust, and the exposure of identities of nearly half the US population.

Another similar case happened in 2021. GoDaddy [6], a major web hosting provider, disclosed a data breach that compromised the information of 1.2 million customers. The breach stemmed from a compromised password, granting attackers access to sensitive account data, including FTP credentials, SSL private keys, and more. Failure stemmed from the fact that some of these credentials were either stored in plaintext or protected with weak encryption. This cryptographic neglect left customers highly vulnerable to phishing schemes and other forms of cyber exploitation. The case underscored how even limited unauthorized access can escalate quickly, leading to full control over websites and exposing vast amounts of user data.

A similar issue came to light around 2019 when security researcher Brian Krebs revealed a serious oversight at Facebook [5]. The company had been storing up to 600 million user passwords in plaintext, with some records dating all the way back to 2012. Although Facebook stated that these passwords were not misused, the incident highlighted the risks of failing to apply even the most basic cryptographic safeguards. The Data Protection Commission (DPC) in Ireland fined Meta 265 million euros after finding that Facebook and Instagram passwords were stored in plaintext. This security lapse occurred because the passwords were unintentionally saved in a readable format within Meta's internal systems [4].

These examples show that inadequate cryptography can lead to long-lasting damage, not only in terms of financial cost but also in the loss of user trust and privacy. As data becomes more central to everyday life, strong encryption practices are essential. This involves not only implementing encryption but also ensuring its robust and correct application to protect data integrity, confidentiality, and authenticity across all digital platforms and systems.

CSE 40567/60567: Computer Security,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

This project is motivated by the need to understand and apply cryptography correctly in real-world scenarios. Our goal is to develop tools and systems that demonstrate the critical role of encryption in creating a safer and more secure digital environment.

2 Survey of Cryptographic Systems and Emerging Research

2.1 Common Cryptosystems in Practice

Modern cryptographic systems fall into two main categories: symmetric encryption, where the same key is used to encrypt and decrypt data, and asymmetric encryption, where separate public and private keys are used. Each type has its own strengths and ideal use cases [10]. In this section, we explore three widely used cryptosystems, AES, RSA, and 3DES, and examine their benefits and limitations in real-world applications [9].

AES (Advanced Encryption Standard) is a symmetric key encryption algorithm that is considered the standard for securing data today. It supports key sizes of 128, 192, and 256 bits and is widely used in applications ranging from securing Wi-Fi networks to protecting sensitive data in financial institutions.

Advantages:

- High efficiency: AES is extremely fast and resource-efficient, making it ideal for encrypting large amounts of data, especially on modern hardware.
- Strong security: It is considered highly secure, with no practical attacks known against AES when used with strong keys and proper implementation.
- Standardized and trusted: AES is approved by the U.S. government and used globally in both public and private sectors.

Tradeoffs:

- Key distribution: Because AES is symmetric, both parties need access to the same secret key, which can be challenging to manage securely over open networks.
- Vulnerability to poor implementation: AES itself is secure, but incorrect use (such as insecure key management or weak modes of operation) can create vulnerabilities.

RSA (Rivest–Shamir–Adleman) is a widely used asymmetric cryptosystem that is typically used for secure key exchange, digital signatures, and encrypting small amounts of data. It relies on the mathematical difficulty of factoring large prime numbers.

Advantages:

- Secure key exchange: RSA allows two parties to establish secure communication without sharing a secret key in advance.
- Digital signatures: It provides authentication and non-repudiation, making it valuable for verifying identities and ensuring message integrity.
- Mature and well-understood: RSA has been studied extensively and is supported by nearly all cryptographic libraries and protocols.

Tradeoffs:

- Slow performance: RSA is computationally intensive and not suitable for encrypting large volumes of data.
- Key length inflation: To maintain security over time, RSA requires increasingly larger key sizes (e.g., 2048-bit or 4096-bit), which further reduces performance.
- Vulnerable to quantum computing: Future quantum algorithms, such as Shor's algorithm, could break RSA by factoring large numbers efficiently.

Triple DES (3DES) is an older symmetric encryption algorithm that applies the DES (Data Encryption Standard) cipher three times to each block of data. It was developed to extend the life of DES after its original 56-bit key was deemed insecure [7].

Advantages:

- Legacy compatibility: 3DES is still supported in older systems and protocols, making it useful in environments with outdated infrastructure.
- Increased security over DES: By applying DES three times with different keys, it significantly improves security compared to its predecessor.

Tradeoffs:

- Poor performance: 3DES is significantly slower than AES and is not suitable for high-throughput applications.
- Limited security lifespan: Even with triple encryption, its effective security is lower than modern algorithms, and it is vulnerable to certain attacks like meet-in-the-middle. Being phased out: Most modern standards, including NIST guidelines, discourage the use of 3DES in favor of AES.

2.2 Emerging Research Topics

As digital systems grow in complexity and scale, new cryptographic challenges continue to arise across various domains. Three key areas currently driving research in cryptography include asynchronous consensus [2] without trusted setup or public-key cryptography, securing IoT devices using blockchain and quantum cryptography [3], and cryptographic techniques in cloud computing. This section focuses on the third topic, which addresses the evolving challenges of securing data in cloud environments.

2.3 Focus Topic: Cryptographic Techniques in Cloud Computing

Cloud computing enables users to access storage, processing power, and various services over the internet without owning or maintaining dedicated hardware. While this model provides scalability, flexibility, and cost efficiency, it also raises pressing concerns related to data privacy, integrity, and access control. The main challenge lies in the fact that users must relinquish direct control over their data, trusting third-party providers to handle sensitive information securely. This creates significant risks, particularly in multi-tenant environments where infrastructure is shared among multiple users, increasing potential exposure to breaches.

One of the key technical challenges in cloud computing is maintaining data confidentiality without sacrificing performance. Strong encryption algorithms are resource-intensive and can hinder the cloud's ability to deliver fast, scalable services—especially for applications involving large datasets or real-time processing. Balancing

robust security with the need for high performance remains a complex problem. Another critical issue is maintaining proper data isolation and access control in multi-tenant systems. Cloud providers must ensure that the data of different users remains strictly separated while also guaranteeing that only authorized parties can access sensitive information. Achieving this level of isolation is challenging because inadequate separation could expose data to unauthorized users or malicious actors.

Emerging cryptographic techniques such as fully homomorphic encryption (FHE) offer a potential solution by enabling computations directly on encrypted data without requiring decryption. This approach protects confidentiality even while processing occurs in the cloud, but FHE remains largely impractical for widespread deployment due to its extreme computational expense and lack of scalability in real-world applications. Conversely, lightweight cryptography provides efficiencies that suit resource-constrained devices, such as those in IoT ecosystems connected to cloud platforms [8]. However, these lightweight methods often come with trade-offs in security strength, making them unsuitable for certain high-risk applications.

To address these challenges, researchers are exploring a combination of established and emerging cryptographic techniques. Elliptic Curve Cryptography (ECC) is identified as a particularly promising method for securing communication within cloud environments. ECC achieves strong levels of security with relatively small key sizes, reducing computational overhead compared to traditional algorithms like RSA. This makes ECC well-suited for cloud systems where performance is critical. The paper also highlights the benefits of using hybrid cryptographic solutions that combine symmetric and asymmetric methods. For example, symmetric algorithms like AES provide the speed necessary for encrypting large amounts of data, while asymmetric techniques are used for secure key exchanges. This hybrid approach leverages the strengths of both methods to achieve a practical balance between security and efficiency.

Additionally, the research explores more novel approaches such as DNA-based cryptography for niche applications, as well as advocating for lightweight encryption techniques in cloud-connected IoT environments where processing resources are limited. These emerging methods reflect the broader effort to align cryptographic technologies with the evolving demands of cloud computing while addressing concerns about data ownership, privacy, and scalability.

As cloud computing continues to expand and underpin critical digital infrastructure, developing robust and efficient cryptographic techniques remains a pressing research priority. Ongoing work in this area aims to provide solutions that allow organizations to fully harness cloud capabilities without compromising the security and integrity of their data.

3 Implemented Solutions

We developed a user-friendly, interactive GUI-based tool in Python that supports multiple encryption algorithms, using the pygame and *pygame_gui* framework [**? ?**]. These tools allowed us to create an interface that lets the user select which algorithm they would like to use and dynamically be able to switch between them.

```
if event.type == pygame_gui.UI_BUTTON_PRESSED:
    # Tab switching
    if event.ui_element == vigenere_button:
        hide_all_panels()
        vigenere_panel_container.show()
    elif event.ui_element == rsa_button:
        hide_all_panels()
        rsa_panel_container.show()

    elif event.ui_element == aes_button:
        hide_all_panels()
        aes_panel_container.show()
    elif event.ui_element == des_button:
        hide_all_panels()
        des_panel_container.show()
```

Figure 1: Panel control logic

Given that every type of algorithm has unique needs, such as 3DES requires 3 keys while AES only requires one, we modularized the interface by creating a dedicated UIPanel for each algorithm, each initialized with the specific UI elements it needed. These panels were all managed by a shared UIManager, and toggled via UIButton to try and imitate a tab-switching behavior. Once a button is switched, the panel corresponding to that algorithm will become visible and the others will be hidden away. Given that initially, each person had been in charge of coding different algorithms and producing their own GUI, the main issue is that although mostly consistent, there are some aspects of the interface that switch, which can make the user experience slightly harder.

To be able to unify all the different algorithms, each cipher panel operates independently but within the same event loop, and UI manager, as to be able to handle events effectively.

3.1 Vigenère Cipher

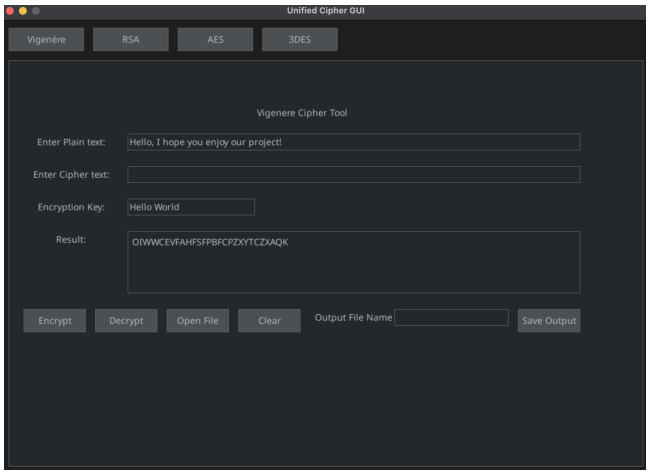

Figure 2: Vigenère GUI

The Vigenère cipher is a polyalphabetic substitution cipher introduced in the 16th century, often considered a historical milestone

in cryptographic systems. It extends the Caesar cipher by using a keyword to shift each letter of the plaintext by varying amounts, thereby complicating frequency analysis. Each character in the plaintext is encrypted by aligning it with a corresponding character in the key, looping the key if it is shorter than the text: each character in the plaintext is aligned with a corresponding character from the keyword. The position of the letter in the alphabet defined by the keyword determines the number of positions the plaintext character is shifted forward. For example, if the letter 'B' (the second letter of the alphabet) appears in the key, the corresponding plaintext letter is shifted two positions. This mechanism results in a ciphertext that is the same length as the original message and does not require padding or any additional structural obfuscation [1].

At the time of its popularization, the Vigenère cipher was regarded as virtually unbreakable and was even dubbed "le chiffre indéchiffrable," or "the indecipherable cipher." However, as cryptanalytic techniques evolved, particularly in the 19th century, the cipher was eventually broken through methods that exploited the statistical properties of natural languages. Techniques such as the Kasiski examination and the Index of Coincidence (IoC) analysis enabled cryptanalysts to deduce the length of the keyword and, subsequently, apply frequency analysis to the separated Caesar ciphers that result from each letter in the keyword. These breakthroughs ultimately exposed the vulnerabilities of the Vigenère cipher and underscored the need for more complex and secure encryption systems.

Despite its eventual fall to more advanced cryptanalysis, the Vigenère cipher remains a historically important and pedagogically valuable cryptographic technique. It illustrates the transition from simple to more complex encryption methods and continues to be studied for its conceptual clarity and its role in the evolution of modern cryptography.

The Vigenère cipher was implemented from scratch using Python, with added functionality to support both encryption and decryption based on user-supplied keys. A significant feature of the implementation was the ability to decrypt ciphertexts without a known key by performing statistical analysis using IoC and chi-squared comparison against expected letter frequency distributions. To make the tool accessible and interactive, a graphical user interface was developed using 'pygame gui'. The interface allowed users to enter plaintext and ciphertext, input or omit a key, and visualize decrypted results alongside the inferred key when applicable.

3.2 RSA

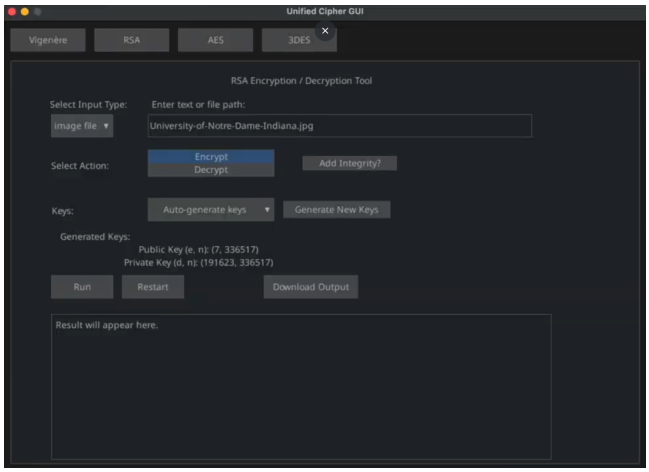

Figure 3: RSA GUI

RSA is a widely used public-key cryptographic algorithm that secures data by leveraging the mathematical difficulty of factoring large integers. It involves two keys: a public key for encryption and a private key for decryption. The security of RSA is based on the computational hardness of the integer factorization problem. Given a large modulus n=pq, where p and q are large primes, it is infeasible to determine p and q from n alone. The typical key length used for secure applications ranges from 2048 to 4096 bits, which offers strong protection against brute-force attacks and ensures robust confidentiality when properly implemented.

For RSA, we implemented an algorithm and GUI (shown in Figure 3) to support encryption and decryption of plain text, binary, text files, and image files. It provides two modes of operation: standard RSA and an extended "integrity" mode. In standard mode, the user either manually inputs or automatically generates a public/private key pair, and the tool uses these keys to perform basic RSA encryption and decryption. To ensure key validity, the program checks that both the public and private keys share the same modulus n, raising errors otherwise. The interface guides the user through entering the components e, d, and n, or generating them via a built-in key generation function.

In integrity mode, we extended RSA to include both sender and receiver key pairs to support authenticated encryption. In this case, data is first signed using the sender's private key and then encrypted with the receiver's public key, adding an authentication layer in addition to confidentiality. During decryption, the receiver first decrypts the message using their private key and then verifies the origin using the sender's public key. The interface dynamically adapts to this mode by exposing additional input fields and toggling visibility based on user selection.

The core processing logic is handled in a dedicated function that reacts to the "Run" button. It parses the user's selections, reads input data, processes key fields, and applies the appropriate RSA routine. For image and file-based input, the tool handles file I/O, converts the contents to bytes, and processes them. The tool provides feedback

in the form of decrypted text or prompts the user to download the resulting files for image content. Additionally, the system includes input validation, error handling, and status messages to guide user interaction and prevent incorrect cryptographic operations.

3.3 3DES (Triple DES)

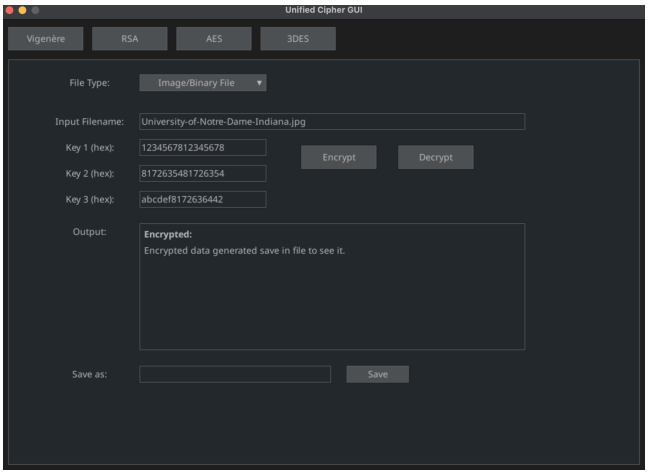

Figure 4: 3DES GUI

The 3DES algorithm is a symmetric-key encryption method that strengthens the original DES by applying the DES algorithm three times in succession. During encryption, the plaintext block is first encrypted with key 1, then decrypted with key 2, and finally encrypted again with key 3. Decryption follows the same logic but in reverse: it starts by decrypting with key 3, then encrypts with key 2, and finishes by decrypting with key 1.

The level of security for this algorithm can vary depending on the relationship between the three keys that the user inputs. The 3-key 3DES mode (where $K1 \neq K2 \neq K3$) provides the highest security and a key length of 168 bits; however, its effective security is closer to 112 bits due to attacks like meet-in-the-middle. The 2-key 3DES mode ($K1 = K3 \neq K2$) reduces the key size to 112 bits; however, since it is susceptible to some chosen-plaintext or known-plaintext attacks, it is designated by NIST to have only 80 bits of security. Although it has a lower security level, it is still useful because it provides backward compatibility while being more secure than DES. Finally, if all three keys are the same ($K1 = K2 = K3$), 3DES is functionally equivalent to single DES, offering no added security and defeating the purpose of using 3DES altogether.

In our implementation, we built the DES algorithm from scratch. We followed the standard process, which includes generating 16 subkeys using specific bit rearrangements (PC-1 and PC-2) and a series of shifts. The user's plain text, is divided into 64-bit blocks and each block goes through an initial rearrangement (IP), then 16 rounds of processing using the Feistel structure, and ends with a final rearrangement (IP-1). We reused this DES process three times for each 3DES operation. To make sure the data fits properly, we added padding to the input, and we used Base64 to make the output readable when needed.

To be able to handle multiple input formats, we used different checks throughout to verify the correct format. For example, cipher text input by the user had to be a valid Base64 string and its length had to be a multiple of 8 bytes, since DES processes data in 64-bit (8-byte) blocks. To handle binary input, we checked that it only contained 0s and 1s and that its total length was a multiple of 8 to ensure each chunk could be properly converted into a byte. If the input didn't meet these size requirements, we raised an error before attempting to encrypt or decrypt, to avoid breaking the algorithm. To handle image files we read the files by bytes and encrypted block-by-block. The GUI dynamically updates based on the user's selected input type, and input validation ensures that all keys are properly formatted hexadecimal strings.

Contrary to what people believe although 3DES performs three DES operations, it is not three times as secure as DES due to vulnerabilities like the meet-in-the-middle attack, which reduces its effective strength significantly. 3DES is also vulnerable to sweet32 attacks and block collision attacks, because of the short block size and using the same key to encrypt large sizes of text.

3.4 AES (Advanced Encryption Standard)

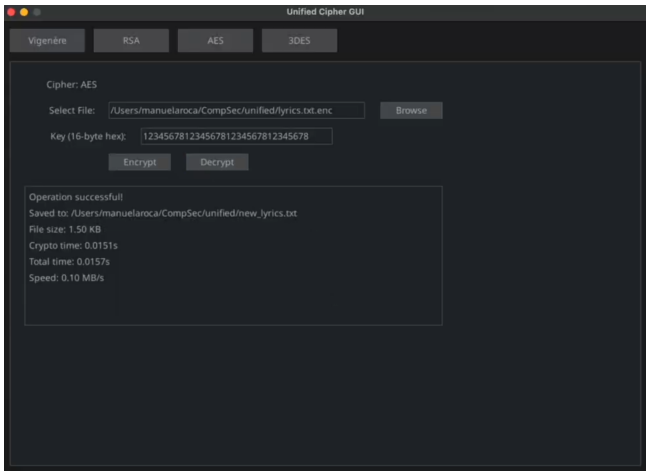

Figure 5: AES GUI

The Advanced Encryption Standard (AES) is a symmetric-key block cipher established by NIST(National Institute of Standards and Technology) in 2001 as a replacement for DES(Data Encryption Standard). AES operates on 128-bit blocks and supports key lengths of 128, 192, or 256 bits, with the number of encryption rounds (10, 12, or 14) varying based on the key size. AES utilize a series of reversible transformations—SubBytes, ShiftRows, MixColumns, and AddRoundKey— in order to achieve confusion and diffusion, making it resistant to both linear and differential cryptanalysis. Unlike DES, AES does not use a Feistel structure but processes entire blocks through substitution-permutation networks. AES is widely regarded as the secure standard for modern applications, currently with no practical brute-force or mathematical attacks against properly implemented AES-256.

In our implementation, AES was integrated into the GUI as a symmetric-key algorithm requiring a single key length of 128 bits.

The interface accepts a hexadecimal key input of exactly 32 characters (16 bytes). The input validation rejects non-hex characters or incorrect lengths. For the text input, plaintext is padded using PKCS#7 to align with AES's 128-bit block size, while binary and file inputs are processed in byte streams. Image files are encrypted block-by-block, preserving metadata to maintain file integrity after decryption.

Our GUI implementation also provides real-time feedback for invalid inputs, such as non-hex keys or improperly sized binary strings. Keys with lengths not 32 characters trigger 'Key must be 16 bytes (32 hex chars)' warnings, and non-hex characters produce 'Invalid key format' errors.For decryption. The system also verifies that the input ciphertext length is a multiple of the block size and validates padding after decryption. Unlike 3DES, AES avoids vulnerabilities linked to small block sizes by using a larger 128-bit block, mitigating risks like collision attacks. While our implementation defaults to ECB mode for simplicity, future iterations could integrate secure modes like CBC(Cipher Block Chaining) or GCM(Galois/Counter Mode) with initialization vectors (IVs) for enhanced confidentiality.

4 Performance Analysis

Analyzing execution time and ciphertext size is critical for analyzing the performance of the cipher algorithms. Execution time tells us how efficiently an algorithm is capable of processing data, a critical component for any application that requires real-time performance and scalability. On the other hand, ciphertext size analyzes the growth of an original input when encrypted, this is essential to understand the impact of encryption on the storage space, transmission bandwidth and overall system efficiency. Both of these metrics will allow us to understand how well each algorithm performs

4.1 Execution time

The performance analysis of the four cryptographic algorithms: Vigenère, RSA, 3DES, and AES, show significant variation in execution time depending on the file size and algorithm used.

Cryptographic Execution Time Analysis						
	Small File		Medium File		Large File	
Algorithm	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
Vigenere	0.00045	0.000431	0.444166	0.506893	54.673503	50.947971
RSA	0.001381	0.001951	0.200382	1.077045	34.585255	119.705563
3DES	0.079189	0.077956	83.76696	83.09318	672.33422	675.879233
AES	0.004887	0.007009	2.989333	3.953309	296.205651	412.689332

Figure 6: Execution Time Analysis Table

As shown in the table above, all algorithms perform relatively quickly when working with small files (1 KB). Vigenère demonstrates fast encryption and decryption times, while 3DES is the slowest. When moving to a medium-sized file (1 MB), the differences in performance become more noticeable. At this stage, RSA emerges as the fastest algorithm for encryption, while Vigenère remains the fastest for decryption. However, the slowest algorithm overall continues to be 3DES. Finally, for large files (100 MB), the same general pattern holds as for medium-size files; RSA maintains the fastest encryption time, Vigenère is the fastest for decryption, and 3DES is consistently the slowest for both encryption and decryption.

Given that Vigenère is a simple polyalphabetic substitution cipher, it makes sense that it's one of the fastest algorithms, throughout all the file sizes. Vigenere is computationally fast due to its simplicity, however because of that it is not as secure as some of the others. What's surprising is RSA is the fastest for encryption for large and medium size files, especially since asymmetric cryptographic algorithms are typically slower than symmetric ones. We assume there are two main reasons for this unexpected result. First, our implementation doesn't use prime numbers large enough to be fully secure, and that significantly reduces the computational load. Second, both 3DES and AES, which we'd expect to be faster, introduce a lot more overhead which just slows down their performance. The overhead would also help explain why 3DES is so slow overall, especially since every 64-bit block has to go through 48 rounds of processing (16 rounds × 3 DES passes) so any type of overhead is quite substantial.

4.2 Cipher-text size

Cipher-Text Performance		
	Size of cipher text (Bytes)	
Algorithm	Small File	Medium File
Vigenere	1024	1048576
RSA	1296	1048848
3DES	1376	1398112
AES	1040	1048592

Figure 7: Cipher-Text Size Performance

We also measured the performance of the algorithms in terms of ciphertext size. The differences across algorithms are also notable as seen in Figure 2. Vigenère produces ciphertexts of the same size as the original files, as it maps characters without adding any padding. AES, despite being a block cipher, showed minimal ciphertext growth, only 16 bytes larger than the original medium size file (1MB), we believe this is due to its 128-bit block size and the use of an efficient padding scheme. 3DES and RSA both produce ciphertexts larger than the original files due to block size alignment and padding. 3DES results in the largest ciphertexts, likely because of its 64-bit block size and triple-layer processing which means that there is more structural overhead added and it will require more frequent padding than other algorithms. RSA generated slightly more overhead, adding 272 bytes for the medium file, again, reflecting the structure padding and formatting required for RSA's block-wise encryption.

4.3 Performance conclusion

The results from the test performed show that there is a balance between security, performance speed, and ciphertext growth. Although the Vigenère algorithm was the fastest overall and had the smallest ciphertext, it is also the least secure by modern standards, since it can be broken due to its repeating key. AES on the other

hand, even though it was shown to be third overall fastest algorithm and the one with the least ciphertext growth after Vigenère is considered the most secure. This shows that algorithm selection really depends on the use case. For example, if you're encrypting a small piece of text, even though AES is the preferred standard, RSA could also be used since it would still be secure and might even run faster in certain cases. The performance of 3DES also showed why it's an algorithm that is not being used anymore as a security algorithm, in fact it was officially banned in 2023 according to Doverunenr. With the highest ciphertext growth and the slowest execution time, and the lack of performance not being compensated by being an unbreakable algorithm such as AES makes it clear why it is no longer recommended for modern use.

5 Group Collaboration and Development Challenges

5.1 Work distribution

To complete this project, each team member contributed significantly to the development, analysis, and presentation of our cryptographic system.

Ziyun focused on implementing the RSA algorithm and exploring the emerging research in the field of cryptography. Additionally, her work included articulating the motivation behind the project, which emphasized the increasing relevance of data security in a digitally interconnected world. Finally, she played a key role in editing the demo video that illustrates how our code works.

Rana was responsible for implementing the Vigenère cipher and took the lead in writing the abstract, conclusion and future work, refining the motivation, and references sections of the paper, providing a clear summary and outlining potential directions for extending this project.

Haotian implemented the AES (Advanced Encryption Standard), ensuring secure and efficient symmetric encryption within the unified interface.

Manuela worked on the implementation of the 3DES (Triple Data Encryption Standard). She also focused on developing a unified user interface that integrates all the encryption algorithms, which required careful management of all four algorithms to maintain their functionality. She also conducted a performance analysis across the algorithms to compare their efficiency and effectiveness under various conditions, and worked on the challenges and solutions descriptions. Furthermore, Manuela collaborated with Ziyun on editing the demo video, ensuring that the final product clearly demonstrates the system's design and functionality.

5.2 Challenges and Solutions

Vigenère was conceptually simple to implement as it depended on simple mapping, however, there were two main difficulties we encountered. The first one is that to maintain the highest level of security the encrypted text removes all the spaces, therefore, when we decrypt the cipher, we are not able to get back those spaces which does affect the integrity of the information as it's not the same. The same happens with any type of another character that is not ASCII, characters like '.', ',', '!', '?' ' ´ ' and more are lost during the encryption and decryption process. The same idea happens when capitalizing letters, to maintain the highest level of safety these are also lost during translation. To solve this problem we store formatting separately in metadata, mapping the special characters and the spaces they would go into, however, this decreases the security level, and therefore, as a group, we decided not to implement this solution. The second challenge faced with the Vigenère algorithm is the implementation of frequency analysis tools like the Index of Coincidence and chi-squared statistics, which depended heavily on having a sufficiently large ciphertext sample. When the input text volume was low, the statistical methods for key recovery often yielded inaccurate or ambiguous results, highlighting the limitations of such techniques in real-world applications. Moreover, the debugging process revealed that even small errors in letter shifting or modular arithmetic could produce unintelligible outputs.

For RSA the biggest challenge was on the key generation and modular arithmetic. We had a lot of trouble trying to find keys that were secure enough but also fast enough for the GUI. In the end, given our CPU's limitations and our main objective; which was to demonstrate how the different algorithms work, we compromised with prime numbers that weren't big enough to be considered cryptographically secure but were enough to demonstrate the principle of RSA. The biggest technical challenge was getting block-wise encryption and decryption to work correctly. Since RSA can't process large files or messages at once, we had to break the input into chunks smaller than the modulus making it tricky to ensure that padding didn't break on short blocks, especially with binary or image data. To solve this, we developed a custom padding scheme that mimicked PKCS-style padding. On decryption, we stripped padding carefully trying to ensure no loss of valid data. Finally, to be able to properly handle different types of inputs from the user, we had to introduce custom handling.

3DES was one of the most time-consuming algorithms to implement since we had to implement DES from scratch. The biggest problem in our 3DES algorithm is the high overhead it has given all the permutations and the 16 rounds for each DES. This made testing 3DES hard, especially for larger files. We tried reducing the overhead as much as possible to address this concern, however, as seen in the performance analysis it is still the slowest algorithm by a lot. One of the biggest challenges when implementing this algorithm was preserving format and handling binary input, because of how delicate the header of a file is, any type of corruption would render (especially image files) unreadable. Since 3DES operates by 64-bit blocks, and there are a lot of permutations, we had to make sure every byte was managed correctly and more importantly that padding was only applied to the actual data and not the metadata. Another, challenge we faced was maintaining the symmetry between encryption and decryption, initially, we had two different functions for each, however, because we needed to jeep byte alignment perfectly we soon realized it was easier to use the same DES function and just reverse the key order, instead of having different ones. Finally, just like RSA, we struggled when trying to get the algorithm to work in the different types of inputs (text, hex, binary string, images, and files), since each of them required different handling, especially when getting a cipher text from the user as it has to meet specific qualifications, to solve this issue we just created different handlers, to transform the data into the correct format so that we could use the DES functions.

For AES, we built our own version from scratch using ECB mode, where each 128-bit block of data is treated like one big number. The hardest part was getting the different transformation steps; SubBytes, ShiftRows, MixColumns, and AddRoundKey, to work correctly. These steps had to be applied in the right order, and we had to constantly convert between binary, hex, and matrices, which just like in 3DEs proved to be trickier than initially thought, and it also made debugging harder. We didn't use CBC mode or add support for IVs or padding, so we had to make sure that all inputs were already the right length (multiples of 16 bytes), we addressed this by enforcing strict input length validation during GUI use and ensuring all blocks conformed to AES expectations. The key expansion step also took some time to get right because we had to generate round keys using the S-box and Rcon values.

```python
def check_key(key):
    key = key.strip() #remove trailing char

    #check for binary key, (if ambiguous case: key is composed of only 0 and 1 assume binary)
    if len(key) == 64:
        for char in key:
            if char not in ['0', '1']:
                break
            else:
                return "binary"

    if len(key) == 16: #check for hex digits
        for char in key:
            if not (char.isdigit() or char.lower() in ['a', 'b', 'c', 'd', 'e', 'f']):
                break
            else:
                return "hex"

    return None #incorrect length (INVALID KEY )
```

Figure 8: 3DES key check

Given that each encryption algorithm required user input, often in different formats and with strict structural rules (e.g., AES requiring a 32-character hexadecimal key), a major challenge was ensuring reliable operation across all cases. Figure demonstrate the key check function implemented for 3DES, it verifies that the key is given in hex or binary, with the correct number of bytes. Users might provide inputs in incorrect formats, invalid lengths, or incompatible types (e.g., binary strings, text, or Base64). To address this, each algorithm was implemented using a Defensive Input Validation approach. This included extensive error checking, try-except blocks, and routines capable of parsing and validating various data types. However, this meant adding computational overhead and increased control flow complexity, since the system had to process extra validation logic for every input case. Despite this, the benefits in reliability and user safety outweighed the costs.

Outside of the logical implementation of the algorithms, our entire group struggled with creating the graphical user interference (GUI) of the codes. Since none of the members had learned how to do a GUI before this project, we had to learn how to use the pygame framework in a short period of time. Especially the hardest part was integrating all four algorithms given that each cryptographic algorithm has different requirements, and the GUI needs to be able to dynamically update depending on the algorithm selected. Therefore, we struggled with modifying the programs which each had its own event handler to unify them, and make sure elements of each algorithm wouldn't interfere with the others.

6 Conclusion and Future Work

Our project successfully designed, implemented and evaluated a multifunctional cryptographic interface that encompasses both classical and modern encryption algorithms: Vigenère Cipher, Triple DES (3DES), Advanced Encryption Standard (AES), and Rivest-Shamir-Adleman (RSA). T. By implementing each algorithm from scratch, we gained valuable hands-on experience that deepened our understanding of their operational principles, cryptographic strengths, and implementation challenges of each algorithm. The custom graphical user interface (GUI), built using the pygame library, enabled users to interact with the algorithms in a user-friendly and intuitive manner. It supported a wide range of input types including plain text, binary data, and full files, effectively simulating real-world encryption scenarios. This flexibility not only enhanced usability but also allowed us to test and observe the algorithms' performance across different data formats.

The inclusion of the Vigenère Cipher served as a valuable pedagogical tool, demonstrating the principles of polyalphabetic substitution and highlighting its susceptibility to frequency analysis, thereby underscoring the advancements in modern cryptography. The implementation of 3DES, AES, and RSA addressed the complexities of contemporary cryptographic practices, including secure padding schemes and byte alignment, particularly when handling diverse data formats such as files and images. Furthermore, the tool's capability to track processing time offered a preliminary basis for comparing the efficiency of the implemented algorithms.

While the project achieved its primary objectives, several avenues for future work have been identified. Firstly, a more rigorous performance evaluation, including benchmarking against established cryptographic libraries and analyzing scalability with larger datasets, would provide a more comprehensive understanding of the tool's efficiency. Secondly, exploring additional cryptographic algorithms, such as elliptic curve cryptography (ECC) or hash functions for data integrity verification, could enhance the tool's functionality and educational value.

From a software engineering perspective, future efforts could focus on refactoring the codebase to improve modularity and maintainability, particularly in the integration of different algorithms within the GUI framework. Addressing the challenges encountered in unifying event handlers and ensuring non-interference between algorithm-specific elements would be a key improvement. Additionally, incorporating more sophisticated key management techniques, such as key derivation functions and secure key storage, would be a valuable extension. Finally, exploring methods to mitigate potential side-channel attacks, although beyond the initial scope of this project, represents an important direction for future research and development.

In conclusion, this project provides a solid foundation for understanding and experimenting with cryptographic systems. The developed tool serves as a valuable educational resource, and the identified future work outlines promising directions for further exploration and enhancement in the realm of cryptography and securing data.

References

[1] Derek C. Brown. 2018. A cryptanalysis of the autokey cipher using the index of coincidence. In *Proceedings of the 2018 ACM Southeast Conference* (Richmond,

Kentucky) *(ACMSE '18)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages. doi:10.1145/3190645.3190679

[2] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. 2024. Asynchronous Consensus without Trusted Setup or Public-Key Cryptography. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 3242–3256. doi:10.1145/3658644.3670327

[3] Shalini Dhar, Ashish Khare, Ashutosh Dhar Dwivedi, and Rajani Singh. 2024. Securing IoT devices: A novel approach using blockchain and quantum cryptography. *Internet of Things* 25 (2024), 101019. doi:10.1016/j.iot.2023.101019

[4] Data Protection Commission (DPC) in Ireland. 2024. Facebook and Instagram passwords were stored in plaintext, Meta fined. (2024). https://www.malwarebytes.com/blog/news/2024/10/facebook-and-instagram-passwords-were-stored-in-plaintext-meta-fined

[5] Brian Krebs. 2019. Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years. (2019). https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/ Accessed: [Insert Date Here].

[6] Nelson Novaes Neto, Stuart Madnick, Anchises Moraes G. De Paula, and Natasha Malara Borges. 2021. Developing a Global Data Breach Database and the Challenges Encountered. *J. Data and Information Quality* 13, 1, Article 3 (Jan. 2021), 33 pages. doi:10.1145/3439873

[7] C Atika Sari, Eko Hari Rachmawanto, and Christanto Antonius Haryanto. 2018. Cryptography triple data encryption standard (3DES) for digital image security. *Scientific Journal of Informatics* 5, 2 (2018), 105–117.

[8] K. Sasikumar and Sivakumar Nagarajan. 2024. Comprehensive Review and Analysis of Cryptography Techniques in Cloud Computing. *IEEE Access* 12 (2024), 52325–52351. doi:10.1109/ACCESS.2024.3385449

[9] Gurpreet Singh. 2013. A study of encryption algorithms (RSA, DES, 3DES and AES) for information security. *International Journal of Computer Applications* 67, 19 (2013).

[10] Muneer Bani Yassein, Shadi Aljawarneh, Ethar Qawasmeh, Wail Mardini, and Yaser Khamayseh. 2017. Comprehensive study of symmetric key and asymmetric key encryption algorithms. In *2017 International Conference on Engineering and Technology (ICET)*. 1–7. doi:10.1109/ICEngTechnol.2017.8308215

[11] Yixin Zou, Abraham H. Mhaidli, Austin McCall, and Florian Schaub. 2018. "I've Got Nothing to Lose: Consumers' Risk Perceptions and Protective Actions after the Equifax Data Breach". In *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*. USENIX Association, 83–102.