PROJECT SYNOPSIS

ON

Secure file transfer using AES and XChaCha20 Encryption

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

CYBER SECURITY

Submitted By:

Name:	PRAGATI RAJ	PRAJJWAL GUPTA
Roll No.	22BECCS28	22BECCS29

Group No. 51

Under the supervision of

Mr. Gaurav Thakur

Assistant professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CENTRAL UNIVERSITY OF JAMMU

Rahya-Suchani (Bagla), District Samba-181143, Jammu (J&K)

1.0 Project Overview

In an era where cyber threats escalate alongside our reliance on cloud storage, secure file sharing has become a cornerstone of digital safety. Current systems, often dependent on traditional encryption methods like AES-GCM, struggle with inefficiencies when handling large files and vulnerabilities such as nonce reuse—a critical flaw that can compromise entire security protocols. These shortcomings expose sensitive data to risks like breaches, unauthorized access, and interception during transfers. To address these gaps, this project proposes a hybrid encryption framework optimized for secure file transfer and sharing. The solution aims to enhance scalability, mitigate nonce-related risks, and ensure robust protection for data confidentiality and integrity, even in high-demand cloud environments.

This approach isn't just theoretical, it's built on standards like NIST's AES [1] and modern improvements like XChaCha20 [2], blending battle-tested reliability with cutting-edge safeguards. First, AES encrypts the actual files—its specialty. Thanks to its hardware-friendly design (and global standardization), it's fast, reliable, and perfect for handling large data loads. But here's where XChaCha20 steps in as the backup: it takes the AES key (the "master key" that unlocks those encrypted files) and locks it down with its own encryption. AES's speed is legendary, but its reliance on flawless nonce usage can be a weak link. XChaCha20's massive 192-bit nonce minimizes that risk, acting like a safety net for AES's stricter requirements. By mixing two different algorithms, we're not just doubling down on security —we're future-proofing. If a flaw ever emerges in one method, the other keeps the system standing. In short: AES ensures speed and compliance; XChaCha20 guards the keys. Together, they create a system that's both tough against today's threats and adaptable for tomorrow's challenges.

Research tells us AES is still the go-to for compliance, but it's not without trade-offs. Without hardware boosts like AES-NI, AES-GCM can strain systems—slowing down data pipelines and hogging CPU power. Worse, its security hinges on never reusing "nonces" (unique codes for each encryption). Slip up just once, and attackers could unravel both privacy and data integrity [3]. On the other hand XChaCha20, unlike AES, it thrives in software—even on devices without fancy hardware, like smartphones or IoT sensors—and its gigantic 192-bit nonce makes accidental reuse nearly impossible [2]. By combining these two algorithms, our framework plays to their strengths. AES handles the heavy lifting for bulk file encryption (keeping regulators happy), while XChaCha20 safeguards the keys. We get a system that dodges AES's pitfalls, stays lightning-fast, and adapts seamlessly to everything

from data centers to smart devices.

This project isn't about chasing theoretical breakthroughs. Instead, it's a practical experiment: How can we combine proven encryption tools in smarter ways to solve today's security headaches? By blending AES and XChaCha20, the framework stays nimble enough to adapt as threats evolve (say, adding dynamic key generation down the line) while keeping performance sharp. For organizations juggling compliance mandates and high-speed data demands—hospitals, financial firms, cloud providers—this could mean security that's both bulletproof and business-friendly.

2.0 Existing System

In secure file sharing, AES-GCM is like a trusted locksmith—it's fast, encrypts data, and checks for tampering all at once. Its hardware-friendly design (think turbocharged chips like AES-NI) makes it a go-to for encrypting massive files. But here's the catch: it relies on never reusing a "nonce" (a unique code for each encryption). Reuse that code even once, and it's like handing hackers a skeleton key—they could crack open encrypted files or forge fake data. A tiny slip-up, a giant security disaster. [8].

Let's understand some challenges in AES.

2.1 Nonce Management Challenges

AES-GCM requires a unique nonce (number used once) for each encryption operation. Reusing a nonce with the same key can lead to severe security vulnerabilities, including the potential compromise of the encrypted data's confidentiality and integrity. The Internet Engineering Task Force (IETF) emphasizes the critical nature of unique nonces, noting that nonce reuse can result in the complete failure of the mode's authenticity, enabling attackers to inject data into the encrypted communication [4].

2.2 Performance Considerations

The performance of AES-GCM is significantly enhanced by hardware acceleration features such as Intel's AES-NI. In environments where such hardware support is unavailable, AES-GCM's performance can degrade, leading to increased CPU usage and reduced throughput. A performance analysis conducted by Red Hat on IPsec implementations revealed that, despite the absence of expensive hardware accelerators or NIC ESP offloads, IPsec over AES-GCM associations emerged as a compelling alternative, delivering high throughput and secure

communication [5].

2.3 Alternative Encryption Methods

To address the limitations of AES-GCM, alternative encryption methods have been explored. ChaCha20-Poly1305 has emerged as a viable option, particularly in environments lacking hardware acceleration. ChaCha20-Poly1305 offers comparable security guarantees to AES-GCM and demonstrates better performance on platforms without specialized hardware support. This makes it particularly suitable for mobile devices and other resource-constrained environments [6].

2.4 Conclusion

While AES-GCM remains a widely adopted standard for secure file transfer, its reliance on unique nonces and hardware acceleration presents challenges in certain scenarios. The necessity for meticulous nonce management and the potential performance bottlenecks in hardware-limited environments highlight the need for alternative or supplementary encryption methods to ensure robust and efficient secure file transfer systems.

3.0 Proposed System

In the context of vulnerabilities associated with AES-GCM, particularly its sensitivity to nonce reuse which can lead to severe security breaches, including the potential recovery of the authentication key and compromise of data integrity [4], this project proposes a hybrid encryption framework that combines the Advanced Encryption Standard (AES) and XChaCha20 algorithms to enhance secure file transfer mechanisms. This layered design has practical advantages. AES, being a block cipher, is highly optimized for performance in high-throughput environments, while XChaCha20, being a stream cipher, performs exceptionally well on software-only platforms, including mobile and IoT devices [10] .This approach aims to mitigate the identified weaknesses by using the strengths of both encryption methods.

The proposed system operates by encrypting the file data using AES, a symmetric encryption algorithm well-established for its efficiency and robustness. Finally, the AES encryption key is secured using XChaCha20, an extended-nonce variant of the ChaCha20 stream cipher. XChaCha20 offers significant advantages, including resistance to nonce-misuse and the ability to handle larger nonce sizes, thus reducing the risk of nonce-related vulnerabilities [7]. By encrypting the AES key with XChaCha20, the system adds an additional layer of security,

ensuring that even if the AES-encrypted data is intercepted, the key remains protected. This dual-encryption strategy not only enhances the confidentiality and integrity of the transmitted files but also addresses the performance limitations observed in environments lacking hardware acceleration. While AES performs optimally on hardware-accelerated platforms, its efficiency can degrade in software-only environments. In contrast, XChaCha20 is designed to perform efficiently across various platforms without relying on specialized hardware, making it suitable for a wide range of devices, including those with limited computational resources [6].

The integration of AES and XChaCha20 in this hybrid framework aims to provide a balanced solution that uses the high-speed encryption capabilities of AES and the robust, nonce-misuse-resistant properties of XChaCha20. This combination ensures that the system is not only secure against common cryptographic attacks but also adaptable to different operational environments, thereby offering a versatile and reliable solution for secure file transfer.

Ultimately, this system does not claim to achieve cryptographic innovation at the theoretical level but rather focuses on practical, secure implementation. It addresses shortcomings of current file encryption systems through a thoughtful, hybridized application of well-established algorithms, as supported by recent comparative cryptographic studies [9] [10]

3.1 Problem Formulation

Secure file transfer is vital in today's interconnected digital landscape, yet existing encryption systems have inherent vulnerabilities that can risk data integrity and confidentiality. One critical challenge is the strict nonce management requirement in AES-GCM, where each encryption operation demands a unique nonce. A single instance of nonce reuse can lead to devastating consequences, such as the leakage of authentication keys and the subsequent compromise of the entire ciphertext [4]. This requirement forces stringent operational controls, and any mismanagement or operational error may expose sensitive data to attackers.

Moreover, traditional hybrid systems that utilize asymmetric encryption (such as RSA) for key encapsulation, while effective in securely distributing keys, suffer from significant computational overhead and potential vulnerabilities with increasing key sizes. As the volume of data and the number of transactions grow, these approaches can become impractical in scenarios lacking specialized hardware acceleration. The escalating complexity of secure key distribution, together with the persistent threat of nonce misuse in AES, necessitates a more robust solution.

The proposed approach seeks to decouple the vulnerabilities of individual algorithms by encrypting file data with AES and then wrapping the AES key using XChaCha20. XChaCha20 provides an extended nonce space, thereby significantly reducing the risks associated with nonce collisions [6]. This layered encryption model does not eliminate the fundamental AES requirement for nonce uniqueness, but it adds an additional barrier that forces an attacker to compromise both layers to succeed. In essence, the hybrid framework aims to reconcile the need for high-speed, standardized encryption with an enhanced protective mechanism that improves the overall resilience of secure file transfer systems. This research addresses the central problem of maintaining secure, efficient, and scalable file encryption in environments where traditional systems are vulnerable to operational errors and evolving cryptanalytic attacks.

3.2 Objectives

1. To conduct a comprehensive literature review on symmetric encryption algorithms, including AES and XChaCha20, and their usage in secure file transfer.

2. To implement a hybrid encryption model integrating AES-256 and XChaCha20 for secure and efficient file transfer.

3. To test and evaluate the system for performance, security, and integrity.

4.0 Features of the Project

The proposed system introduces a hybrid encryption framework that offers several key features designed to strengthen data security and improve performance in secure file sharing. The system encrypts file content using AES, leveraging its well-established efficiency and hardware acceleration capabilities, and then secures the AES encryption key by encrypting it with XChaCha20. This dual-layer approach improves overall security by separating the data encryption process from key protection, thereby mitigating the adverse effects of nonce reuse vulnerabilities in AES-based modes such as GCM [4]. By employing XChaCha20 for key wrapping, the system takes advantage of its extended 192-bit nonce, which dramatically reduces the risk of nonce collisions—a common vulnerability in traditional AES implementations [6].

In addition to enhancing cryptographic security through algorithmic diversity, the design emphasizes modularity and adaptability. This modularity allows future enhancements such as the integration of dynamic key generation techniques without requiring a complete redesign of the system architecture. Client-side encryption is another prominent feature, ensuring that

files are encrypted on the user's device before transmission, which minimizes exposure to server-side attacks and data breaches. The system is also designed to support parallel processing, which can significantly reduce encryption and decryption latency when handling large files or high volumes of data. This feature is particularly valuable in environments where processing speed and low latency are critical for maintaining service quality.

Collectively, these features address both the operational and security challenges inherent in current file encryption systems. The combination of AES for high-throughput data encryption and XChaCha20 for robust key protection offers a balanced and scalable solution for secure file transfer, making it suitable for applications ranging from cloud storage services to IoT deployments [8].

5.0 Proposed Methodology

In our research, we follow an experimental, multi-phase approach that begins with an extensive literature review, continues through system design and development, and ends with rigorous performance and security evaluations. First, we will do a comprehensive review of existing secure file transfer solutions, with particular emphasis on hybrid encryption models. In this phase our aim is to identify the strengths and limitations of current systems—specifically those relying solely on symmetric encryption, such as AES-GCM, and those employing hybrid schemes with asymmetric key encapsulation. The insights gathered will be used to justify the design of our dual-layer approach, where the file is encrypted using AES and the AES key is secured with XChaCha20, thereby addressing critical concerns such as nonce reuse vulnerabilities inherent in AES-GCM [4],[6].

Based on the literature findings, the system architecture will be designed to decouple data encryption from key protection. A prototype will be developed using high-level programming languages like Python and Java, employing well-established cryptographic libraries—such as PyCryptodome for AES and PyNaCl for XChaCha20—to ensure both reliability and security. The development phase will emphasize modularity, allowing individual components of the encryption process to be refined independently. In this design, files are first processed in a streaming mode using AES for high-throughput encryption, and the resulting AES key is then encrypted using XChaCha20, taking advantage of its extended nonce capabilities to mitigate nonce collision risks.

Subsequently, we will do a series of benchmark tests to measure key performance indicators such as encryption/decryption times, throughput, and resource utilization across varied file

sizes. These evaluations will be performed under conditions that mimic real-world scenarios, ensuring that the proposed system can operate reliably even in high-throughput and resource-constrained environments. Iterative feedback from testing will be used to fine-tune the system, ensuring that it meets both efficiency and stringent security requirements before final deployment.

6.0 Project Planning

Name of the Activity	Date of Completion	Deliverables	Name of Team member
Literature review	May 31h, 2025	Review report	Pragati
Research challenges and Problem formulation	June 15th, 2025	Use-case analysis	Pragati
Implementation of AES file encryption module	July 30th, 2025	AES Encryption code	Prajjwal
Implementation of XChaCha20 key encryption module	September 15th, 2025	XChaCha20 key encryption module	Pragati
Integration and testing of hybrid module	October 5th, 2025	Hybrid core module	Prajjwal
Performance and security evaluation	October 20th, 2025	Security test report	Prajjwal
Documentation and report writing	November 20th, 2025	Final project report	Pragati

References

[1]J. Daemen and V. Rijmen, The design of Rijndael : AES - the advanced encryption standard ; with 17 tables. Berlin: Springer, 2002.

[2]Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," www.rfc-editor.org, May 2015, doi: https://doi.org/10.17487/RFC7539.

[3] Disadvantage AES-GCM. "Disadvantage AES-GCM." Cryptography Stack Exchange, 31 July 2014, crypto.stackexchange.com/questions/18420/disadvantage-aes-gcm? Accessed 10 Apr. 2025.

[4]A. Choudhury, D. McGrew, and J. Salowey, "AES Galois Counter Mode (GCM) Cipher Suites for TLS," Aug. 2008, doi: https://doi.org/10.17487/rfc5288.

[5]"IPsec performance on Red Hat Enterprise Linux 9: A performance analysis of AES-GCM," Redhat.com, 2024. https://www.redhat.com/en/blog/ipsec-performance-red-hat-enterprise-linux-9-performance-analysis-aes-gcm (accessed Apr. 10, 2025).

[6]"ChaCha20-Poly1305," Wikipedia, Feb. 15, 2023. https://en.wikipedia.org/wiki/ChaCha20-Poly1305

[7]Wikipedia Contributors, "AES-GCM-SIV," Wikipedia, Jan. 08, 2025.

[8]P. V. Maitri and A. Verma, "Secure file storage in cloud computing using hybrid cryptography algorithm," 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Mar. 2016, doi: https://doi.org/10.1109/wispnet.2016.7566416.

[9]Rein Smedinga and M. Biehl, "17th SC@RUG 2020 proceedings 2019-2020," the University of Groningen research portal, May 2020, doi: https://hdl.handle.net/11370/058fd2eb-b3f2-4a59-9a2b-9747a9e765a2.

[10]V. R. Kebande, "Extended-Chacha20 Stream Cipher With Enhanced Quarter Round Function," IEEE Access, vol. 11, pp. 114220–114237, 2023, doi: https://doi.org/10.1109/access.2023.3324612.

(Signature) **(Signature)**
Team Leader **(Project Guide)**

Date: _____