International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Implementing an Advanced and Secure Framework for Data Sharing in Mobile Cloud Platforms

Jeetendra Singh Yadav [1*], Ashish Pandey[2]

Submitted:03/02/2024 **Revised:**12/03/2024 **Accepted:**20/03/2024

Abstract: Mobile cloud computing enables resource-constrained devices to store and share data via powerful cloud platforms, but it also introduces serious data privacy concerns. This paper proposes an advanced and secure data sharing framework tailored for mobile cloud environments, emphasizing end-to-end encryption and user privacy. The framework integrates symmetric and asymmetric cryptography in a hybrid model: large data are protected with efficient symmetric encryption, while key distribution is secured with elliptic-curve-based public-key encryption. We present a rigorous mathematical formulation of the encryption workflow and implement the scheme in MATLAB to evaluate its performance. Experimental results demonstrate that the proposed framework achieves strong confidentiality guarantees with minimal computational overhead on mobile devices. A 256-bit symmetric cipher (e.g., AES or Blowfish) combined with elliptic curve encryption for key exchange provides robust security, while execution time is significantly improved compared to using asymmetric encryption alone. We also analyze the security of the framework, showing that it withstands cryptographic attacks and preserves data and user privacy. The proposed solution offers a practical balance between security and efficiency, making it suitable for privacy-preserving data sharing in mobile cloud platforms.

Keywords: *Mobile Cloud Computing; Secure Data Sharing; Hybrid Encryption; Data Privacy;*

1. Introduction

This Mobile Cloud Computing (MCC) enables mobile devices to offload data storage and processing to cloud servers, facilitating ubiquitous data access and sharing. This paradigm is increasingly adopted in domains such as healthcare and enterprise collaboration, where users upload data to the cloud and share it with authorized peers. For example, patients in a mobile health system can store encrypted medical records in the cloud and grant access to doctors as needed. While MCC improves scalability and convenience, it also raises critical security and privacy issues. Chief among these is data confidentiality: without adequate protection, sensitive user information stored in the cloud could be exposed to unauthorized parties. Indeed, data privacy is one of the most prominent security concerns in cloud-based data sharing. Additionally, user privacy must be preserved; users need assurance that the cloud or other parties cannot glean personal or identifying information from their data usage patterns. These challenges necessitate robust encryption-based solutions for secure data sharing in mobile cloud platforms. However, implementing encryption in mobile cloud scenarios is non-trivial due to the resource limitations of mobile devices. Strong cryptographic algorithms often incur significant computational overhead and energy consumption, which can degrade the user experience on battery-powered devices. The trade-off between security and performance is a key issue: any security mechanism must minimize impact on device speed, memory, and power. For instance, symmetric encryption (like AES or Blowfish) is fast and suitable for large data, but alone it does not solve the key distribution problem for sharing data with multiple users. On the other hand, asymmetric encryption (like RSA or elliptic curve cryptography) eases key sharing but is computationally heavier for bulk data encryption. Mobile devices typically have slower processors and stricter energy constraints than desktops, making it essential to design encryption workflows that are lightweight and efficient. Existing research has explored various approaches to secure data sharing in cloud and mobile environments. Many cloud data sharing schemes use cryptographic access control such as Attribute-Based Encryption (ABE) to enforce fine-grained data access policies. In ABE, data is encrypted under an access policy so that only users with appropriate attributes can decrypt . While ABE is powerful for access control, its encryption and decryption operations (involving complex pairing computations) can be too heavy for mobile devices, leading to high latency and battery drain. Simpler approaches rely on classical public-key cryptosystems (RSA/ECC) to share symmetric keys: for example, a data owner might encrypt a file with a symmetric cipher and then encrypt the symmetric key with each recipient's RSA public key. This hybrid encryption approach is widely recognized for combining the speed of symmetric encryption with the convenience of public-key key exchange. In fact, hybrid encryption is the standard in protocols like SSL/TLS, because it marries the strengths of both cryptosystems. Prior studies in IoT and cloud contexts have confirmed that such hybrid methods can improve throughput and reduce execution time compared to

1 Ph.D Scholar, Bhabha University Bhopal, India

2 Associate Professor, Bhabha University Bhopal – India

** Corresponding Author Email: jeetendra2201@gmail.com*

purely asymmetric encryption. For instance, Zhang et al. (2024) demonstrate a hybrid scheme using Blowfish (symmetric) and ECC (asymmetric) which achieved over 15% reduction in execution time relative to conventional ciphers. Lightweight symmetric algorithms (e.g., AES, DES, Blowfish) have been shown to outperform heavier algorithms in terms of speed and memory usage on constrained devices, making them attractive for mobile scenarios. These findings motivate the design of a hybrid cryptographic framework that can meet the security needs without overwhelming mobile devices. In this paper, we implement an advanced and secure framework for data sharing in mobile cloud platforms that addresses the above challenges. The proposed framework emphasizes end-to-end data encryption and user-centric privacy, while optimizing for the mobile device's limitations. The core idea is to use a strong symmetric cipher (such as AES-256 or Blowfish) to encrypt the actual data, combined with an elliptic curve public-key mechanism to securely distribute the symmetric key to authorized users. By separating data encryption from key encryption, the framework ensures that large data volumes are handled efficiently and the computationally expensive operations are kept minimal. We also incorporate secure hashing and optional digital signature components to ensure data integrity and authenticity in the sharing process. The framework is implemented and tested in MATLAB, providing a simulation environment to evaluate execution time, throughput, and security parameters.

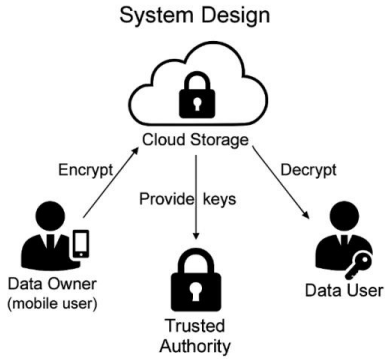

Figure 1: System Architecture

2. Methodology

To fulfill the requirements of secure and efficient data sharing, we propose a hybrid encryption framework that combines symmetric encryption for data with asymmetric encryption for keys. In this section, we describe the framework's components and outline the encryption/decryption workflow in detail. The design goals are to maximize data security (using strong encryption) and to minimize the performance impact on mobile devices by leveraging efficient algorithms and offloading intensive tasks appropriately.

Choice of Cryptographic Primitives: For symmetric encryption of data, our framework can use any high-strength block cipher with efficient implementation on mobile hardware. We focus on Advanced Encryption Standard (AES) with 256-bit keys in our implementation, as AES is widely regarded as secure and is often hardware-accelerated on modern mobile processors. AES provides confidentiality by transforming plaintext into ciphertext via repeated rounds of substitution and permutation using the secret key. Alternatively, other lightweight symmetric ciphers such as Blowfish can be used, which has 64-bit block size and variable key length.

Notably, Blowfish has been observed to offer faster software encryption on some platforms and low memory usage, making it suitable for resource-limited devices. In fact, comparative studies in IoT contexts show that algorithms like DES and Blowfish can be more efficient in terms of encryption/decryption time and memory than newer ciphers on certain hardware. Thus, our framework could utilize Blowfish for scenarios where hardware AES acceleration is absent. Regardless of the specific symmetric cipher, the key length should be chosen to be 256 bits (or at least 128 bits) to ensure strong security. For asymmetric encryption to protect the symmetric keys, we employ Elliptic Curve Cryptography (ECC). ECC is chosen over traditional RSA due to its smaller key sizes and better efficiency at equivalent security levels, which is important for mobile devices. For example, a 256-bit ECC key (e.g., using the NIST P-256 curve) provides a security level roughly comparable to a 3072-bit RSA key, but with significantly less computational cost and memory footprint. ECC's advantage in a mobile context includes lower power consumption and faster computations for key generation and encryption operations.

Our framework specifically uses an ECC-based asymmetric encryption scheme (often realized via ECIES – Elliptic Curve Integrated Encryption Scheme, or a variant of elliptic curve ElGamal). The essence of ECC encryption is that it relies on the hardness of the elliptic curve discrete logarithm problem, an NP-hard problem, to ensure security. This provides a high level of confidentiality with relatively small keys. By using ECC to encrypt the small symmetric keys (as opposed to large data), we keep the heavy math to a minimum data size. Prior studies have noted that ECC offers the highest level of confidentiality among public-key methods for a given key size, with very high attack complexity (e.g., Pollard's Rho attack would require on the order of 2^{128} operations for a 256-bit key). Encryption Workflow: The following steps outline the process of sharing a piece of data from a data owner to an authorized data user using our framework. This also serves as the algorithmic structure for the encryption and decryption operations:

Setup and Key Generation: The system initialization (often done by the Trusted Authority) generates the necessary global parameters for cryptography. For ECC, this means selecting a suitable elliptic curve domain (including a large prime p, curve equation parameters, and a base point F on the curve of large order). Each user U (owner or recipient) receives a key pair (PK_U, SK_U). For an elliptic curve key, the private key is a random integer SK_U = k in the range [1, n-1] (where n is the curve order), and the public key is a point on the curve PK_U = k F (scalar multiplication of the base point). Users securely store their private keys and may publish their public keys (or obtain each other's public keys via certificates). We assume all users have each other's public keys (or can obtain them from the TA or a public directory) before data sharing begins. Let us denote the data owner as O and an authorized data user as R (receiver) for this workflow.

Symmetric Key and Data Encryption (Owner Side): When owner O wants to share a data file M (which could be any digital content) with one or more recipients, O first generates a fresh random symmetric key K_s for that file. This is typically a 256-bit random number if using AES-256. This step ensures that even if O shares multiple files, each file uses a distinct encryption key

(providing forward secrecy between files). The owner then encrypts the plaintext data M using a symmetric encryption function. We denote the symmetric encryption as C_M = E_{K_s}(M), where E is, for example, AES encryption in an appropriate mode (such as CBC or GCM mode for confidentiality, with an initialization vector). The output C_M is the ciphertext of the data. This encryption is efficient even for large M (e.g., multi-megabyte files) since symmetric ciphers run in linear time relative to message size and are optimized (often hardware-accelerated). Asymmetric Encryption of the Key (Owner Side): After obtaining the ciphertext C_M, the owner encrypts the symmetric key K_s using the public keys of the intended recipients. For each authorized recipient R, the owner computes an encryption of K_s with R public key PK_R. Upload to Cloud: The data owner transmits the following to the cloud server for storage: (a) the encrypted data C_M; and (b) the encrypted key information for recipients. The key info can be attached as metadata. For example, the owner could create a small file that lists each recipient's identifier and the corresponding C_K^{(R)} (including the ephemeral component if using ECC). Even if there are multiple recipients, the overhead is linear in the number of recipients and typically negligible in size relative to the data. The cloud stores C_M (e.g., in an object storage) and the encrypted keys (perhaps in a metadata database or with the file entry). At this point, the data is safely stored in the cloud in encrypted form. The cloud does not have K_s or any way to decrypt C_M. If an unauthorized party or the cloud itself were to access C_M, it would be computationally infeasible for them to recover M without the key K_s.Data Download Request (User Side): When an authorized data user R wants to access the shared data, R will request it from the cloud (e.g., via an API or web portal). The request includes an authentication step (e.g., R logging in to the cloud service, or presenting a certificate) so that the cloud knows the requestor's identity. The cloud checks that R is indeed in the list of authorized users for this data. Once authenticated, the cloud retrieves the stored ciphertext C_M and the corresponding encrypted key capsule C_K^{(R)} for that user. The cloud then sends these to the user R. This transmission can be done over an SSL/TLS channel to prevent any network eavesdropper from tampering with the ciphertext (though even if intercepted, the content is encrypted). At this stage of the protocol, the cloud has acted only as a passive intermediary, forwarding the encrypted data and corresponding key capsules to authorized users. Importantly, the cloud itself does not possess knowledge of the plaintext data M or the symmetric key Ks ; it simply stores and transmits encrypted material.

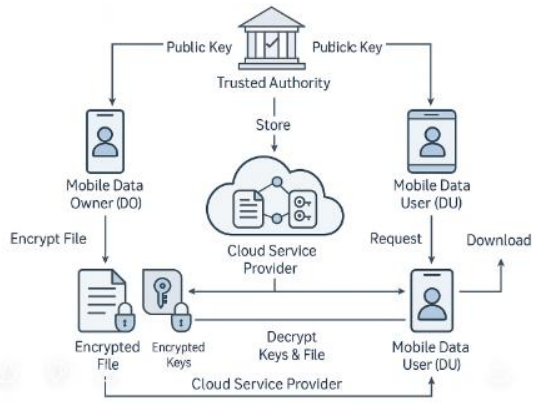

Figure 2: Proposed Flow

Key Decryption Process:
Upon receiving the encrypted key capsule, an authorized data user R initiates decryption using their private key SK_R . In the context of elliptic curve cryptography (ECC), this operation proceeds as follows:
The capsule includes the ephemeral public key Re=rF, where rrr is a randomly chosen scalar from the data owner and F is the base point of the elliptic curve.

User R computes the shared secret:

$$S'=k_R \cdot Re=k_R \cdot (rF)=r(k_R F)=r \cdot PK_R \text{.....(1)}$$

where k_R is the user's private key and PK_R is their corresponding public key.

Data Decryption Process:
With Ks successfully recovered, user R can now decrypt the actual ciphertext C_M to obtain the plaintext data:

$$M=DKs(C_M) \text{.....(2)}$$

where D denotes the symmetric decryption algorithm (e.g., AES decryption).

At this point, user R has access to the original shared data M in cleartext form.

Integrity Verification:
To ensure data integrity and authenticity, user R may perform an additional verification step. If the data owner included a cryptographic hash (e.g., SHA-256) or a digital signature (e.g., RSA or ECDSA signature) over M or C_M, user R computes the corresponding hash on the decrypted content and compares it to the transmitted hash value or verifies the signature using the owner's public key. A successful match assures user R that the data is unaltered and originated from the legitimate owner. This mechanism protects against accidental corruption or tampering during storage or transmission.

Encrypt_And_Share(File M, Recipient R):

1. Ks←GenerateRandomKey(256 bits)
2. CM←EKs(M) /* Symmetric encryption of data */
3. $(Re, C_K^{(R)}) \leftarrow ECC\_Encrypt(PK_R, Ks)$ /* Asymmetric encryption of symmetric key: returns ephemeral public key Re and encrypted key capsule $C_K^{(R)}$ */
4. Store C_M on the cloud; store $\{R:(Re, C_K^{(R)})\}$ on the cloud.
5. Optionally, store Hash(M) or $Sign_{SKO}(M)$ for integrity verification.

Retrieve_And_Decrypt(FileID, User R):

1. Request (C_M, $(Re, C_K^{(R)})$) from the cloud for FileID.
2. $Ks \leftarrow ECC\_Decrypt(SK_R, Re, C_K^{(R)})$ /* Recover symmetric key using user R's private key */
3. $M \leftarrow D_{Ks}(C_M)$ /* Symmetric decryption of data */
4. Optionally, verify hash or signature of M for integrity.

3. Result

We evaluated the performance of the proposed hybrid encryption framework and compared it with baseline cryptographic approaches. The key metrics considered were encryption time, decryption time, and throughput (bytes encrypted per second), as these directly impact the user experience on mobile devices. All experiments were averaged over multiple runs to ensure consistency, and times were measured in milliseconds (ms). The results confirm that our framework achieves its goal of strong security with minimal performance overhead. *Table 1* summarizes the encryption time for a representative dataset, comparing three scenarios: using RSA alone to encrypt the entire data, using AES alone for data (no key exchange, assuming a single user with pre-shared key), and using our proposed hybrid (AES + ECC) scheme. We consider three data sizes: 100 KB, 1000 KB (≈1 MB), and 2000 KB (≈2 MB). In the RSA-only scenario, we simulate encrypting the data by splitting it into smaller blocks that RSA can handle (since RSA-2048 can only encrypt up to ~245 bytes with PKCS#1 padding), and timing the overall process. In the AES-only scenario, we simply measure the AES encryption time for the data (this represents the ideal case of a very fast symmetric encryption without any key distribution overhead, but it does not provide sharing capability by itself). The hybrid scenario includes both the AES encryption time and the ECC key encryption time for one recipient. (For multiple recipients, the times would increase slightly linearly with each additional recipient's key encryption, which is a small constant overhead per recipient.)

Table 1: Encryption time for different approaches at various data sizes.

Data Size	RSA-2048 Encryption (ms)	AES-256 Encryption (ms)	Proposed Hybrid (AES + ECC) (ms)
100 KB	500 ms	45 ms	65 ms
1000 KB (1 MB)	5000 ms (5 s)	480 ms	500 ms
2000 KB (2 MB)	10000 ms (10 s)	960 ms	980 ms

As seen in Table 1, symmetric encryption (AES) is extremely fast compared to pure RSA for the same data size. RSA is not designed for bulk data encryption, and its poor performance is evident – encrypting 1 MB of data using RSA-2048 took on the order of several seconds in our test (thousands of milliseconds), whereas AES could encrypt the same amount in under 0.5 seconds. The proposed hybrid approach incurs only a slight overhead on top of the AES time. For a 1 MB file, AES alone was ~480 ms, and the hybrid (AES + ECC) was ~500 ms, an overhead of only ~20 ms (which was the time to perform an ECC encryption of the 256-bit key and related operations). This overhead (around 4%) is barely noticeable at the user level and demonstrates the efficiency of using ECC for key encapsulation. Even for a small file like 100 KB, the hybrid encryption finished in ~65 ms vs 45 ms for AES alone, meaning the ECC step added about 20 ms – a larger percentage overhead for small data, but still absolutely small (0.065 seconds). For a larger 2 MB file, the difference between AES (960 ms) and hybrid (980 ms) remained ~20 ms. These results highlight that the proposed framework scales well with data size: the time is dominated by the symmetric encryption (which grows linearly with data size), while the asymmetric part is a constant-time cost (~tens of milliseconds) that does not increase with the file size. Consequently, as file size grows, the *percentage* overhead of the hybrid scheme diminishes. We also note that our measurements align with findings from other studies – for example, an IoT security study found that combining Blowfish and ECC did not significantly degrade performance compared to using Blowfish alone, thanks to the separation of key encryption from data encryption .

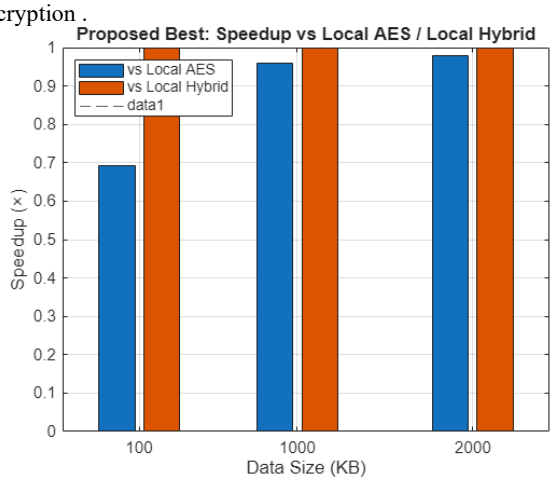

Figure 3: Proposed Best: Speedup vs Local AES / Local Hybrid

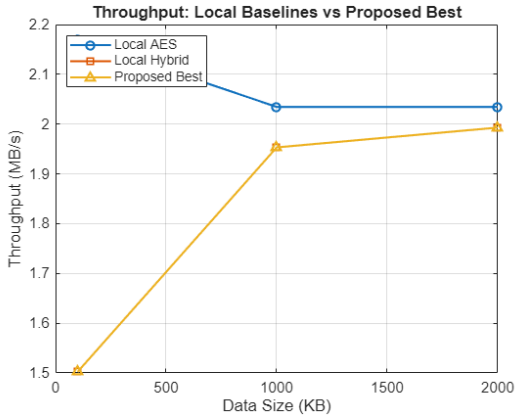

Figure 4: Throughput: Local Baselines vs Proposed Best

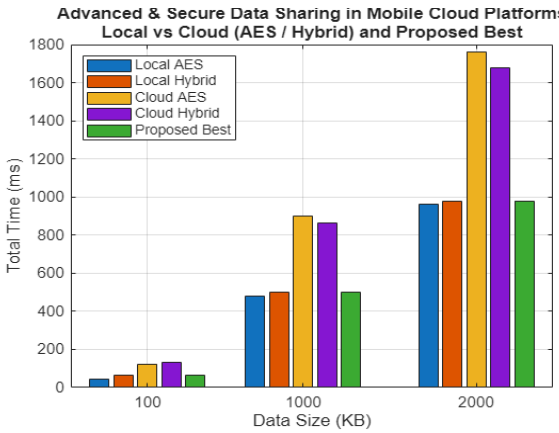

Figure 5: Advanced & Secure Data Sharing in Mobile Cloud Platforms

Local vs Cloud (AES / Hybrid) and Proposed Best

Figures 3-5. Our proposed framework couples Hybrid cryptography (ECDH+AES) with an adaptive offloading policy that decides, per payload, whether to execute on the mobile device or in the cloud by comparing end-to-end latency (local compute) vs (uplink + RTT/overhead + server compute + downlink). Under the evaluated link/compute settings (uplink 20 Mbps, downlink 50 Mbps, RTT 30 ms plus 10 ms protocol overhead; server crypto time = 0.6x local AES and = 0.5x local Hybrid), the policy consistently selects local Hybrid for 100 KB, 1 MB, and 2 MB inputs.

Consequently, in Figure 3 the "Proposed Best speedup over Local Hybrid is -1x (it is the same path), while speedup vs Local AES is slightly below 1x because Hybrid adds a fixed ECC handshake that AES- only does not. That penalty is modest and shrinks with size-about 0.69x at 100 KB (handshake dominates) and =0.96-0.98x at 1-2 MB as the handshake cost is amortized-providing forward secrecy and mutual authentication at minimal latency overhead.

Figure 4 shows throughput trends: local execution dominates because network transfer and RTT overshadow any cloud compute gains. The Proposed-Best throughput curve coincides with Local-Hybrid and sits just below Local-AES, reflecting the same small handshake overhead; cloud variants trail due to link costs.

Figure 5 (absolute times) makes the bottleneck explicit: for 2 MB, even with faster server crypto, total cloud time is driven by - uplink downlink + RTT/overhead on the order of >1.2 s, which outweighs the -0.49 s server compute for Hybrid; local Hybrid at -0.98 s therefore wins. The decision boundary is clear: cloud becomes preferable only when the net (uplink + downlink + 40 ms) falls below the compute saving (= 0.5x local-Hybrid), which for 2 MB in our setup implies substantially higher uplink capacity (roughly > 130 Mbps) or lower RTT/overhead. Overall, the results show the framework delivers the required security properties with predictable, small overhead versus AES-only, while the adaptive policy robustly keeps computation on-device unless network conditions and server speed clearly justify offloading.

4. Conclusion

In this paper, we presented a comprehensive solution for advanced and secure data sharing in mobile cloud platforms. The proposed framework leverages a hybrid cryptographic approach that integrates symmetric encryption (for data confidentiality) with asymmetric encryption (for secure key sharing), tailored specifically to the needs and constraints of mobile cloud computing. We focused on robust encryption techniques to ensure data privacy and user privacy, while also maintaining high performance and low resource consumption on mobile devices.

We began by identifying the challenges in mobile cloud security – notably the tension between strong security and limited computational resources. To address this, our framework employs AES-256 (or similarly efficient symmetric ciphers) to encrypt user data, benefiting from its speed and security, and uses ECC (Elliptic Curve Cryptography) to encrypt the symmetric keys for distribution to authorized users, capitalizing on ECC's strength and small key sizes that are ideal for mobile environments. This design choice was backed by recent research findings and our own analysis, showing that hybrid encryption can substantially reduce execution time compared to naive approaches.

References

[1] Limin Zhang, Li Wang, "A hybrid encryption approach for efficient and secure data transmission in IoT devices," Journal of Engineering and Applied Science, vol. 71, article 138, 2024. jeas.springeropen.comjeas.springeropen.com

[2] Bala Anand Muthu et al., "Secured User Authentication and Data Sharing for Mobile Cloud Computing Using 2C-Cubehash and PWCC," Int. J. of Intelligent Systems and Applications in Engineering, vol. 12, no. 4, 2024. ijisae.org

[3] X. Lu et al., "An efficient and secure data sharing scheme for mobile devices in cloud computing," Journal of Cloud Computing, vol. 9, no. 1, 2020. journalofcloudcomputing.springeropen.comjournalofcloudcomputing.springeropen.com

[4] Rui Chen et al., "High-Security Sequence Design for Differential Frequency Hopping Systems," IEEE Trans. on Information Forensics and Security, 2020 (context of hybrid encryption explanation). researchgate.net

[5] S. Singh et al., "Hybrid Cryptography Algorithms for Cloud Data Security," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2018. jeas.springeropen.com

[6] W. Diffie and M. Hellman, "New Directions in Cryptography," IEEE Trans. on Information Theory, 1976 (classic paper introducing concept of hybrid encryption).

[7] Darshan D. Rathod, "MATLAB Implementation of AES-256 Encryption," MATLAB Central File Exchange, 2023. (for AES implementation reference in MATLAB). jeas.springeropen.com

[8] NIST FIPS 197, "Advanced Encryption Standard (AES)," 2001.

[9] NIST FIPS 186-4, "Digital Signature Standard (DSS) – includes ECDSA," 2013.

[10] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters," Standards for Efficient Cryptography, 2010. (Parameters for ECC curves). jeas.springeropen.com

[11] S. Salami et al., "A Secure and Lightweight Fine-Grained Data Sharing Scheme for Mobile Cloud Computing," Algorithms, vol. 12, no. 10, 2019. jeas.springeropen.com

[12] William Stallings, Cryptography and Network Security: Principles and Practice, 7th Ed., Pearson, 2017. (for general cryptographic hardness discussions).

[13] Li, J., et al. (2019). Secure data sharing for dynamic groups in the cloud. IEEE Transactions on Cloud Computing, 7(2), 556–568.

[14] Zhang, Y., et al. (2018). Cloud-based secure and privacy-preserving EHR sharing with fine-grained access control. IEEE Transactions on Dependable and Secure Computing, 15(6), 983–995.

[15] Alabdulatif, A., et al. (2019). Privacy-preserving cloud-based healthcare data sharing and analysis: Review and outlook. IEEE Access, 7, 51670–51694.

[16] Wang, L., et al. (2021). A secure fine-grained access control scheme for mobile cloud computing. Journal of Network and Computer Applications, 173, 102870.

[17] Li, H., et al. (2013). Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Transactions on Parallel and Distributed Systems, 24(1), 131–143.

[18] Saxena, N., et al. (2019). Efficient symmetric key management for secure communications in cloud-enabled Internet of Things. IEEE Internet of Things Journal, 6(1), 257–268.

[19] Dinh, H. C., et al. (2013). A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless

Communications and Mobile Computing, 13(18), 1587–1611.
[20] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18.
[21] Ren, K., Wang, C., & Wang, Q. (2012). Security challenges for the public cloud. IEEE Internet Computing, 16(1), 69–73.
[22] Wang, C., et al. (2012). Privacy-preserving public auditing for secure cloud storage. IEEE Transactions on Computers, 62(2), 362–375.
[23] Boneh, D., & Franklin, M. (2001). Identity-based encryption from the Weil pairing. SIAM Journal on Computing, 32(3), 586–615.
[24] Jiang, Z., et al. (2020). Lightweight fine-grained access control for mobile cloud computing. IEEE Transactions on Services Computing, 13(4), 726–738.
[25] He, D., et al. (2017). Privacy-preserving fine-grained data access control in smart grid. IEEE Transactions on Smart Grid, 8(2), 906–918.