Secure Sharing of Client-Encrypted Personal Health Records Via Proxy Re-Encryption

D.S.V Geethika[1], Dr. Srinivas Rayudu[2]
[1]*M. Tech Student, Dept. of CSE, Nadimpalli Satyanarayana Raju Institute of Technology, Visakhapatnam*
[2]*Professor, Dept. of CSE, Nadimpalli Satyanarayana Raju Institute of Technology, Visakhapatnam*

***Abstract*—Cloud-based health sharing systems are essential for modern e-health and medical collaboration, yet they remain highly sensitive to privacy leakage methods, such as untrusted cloud monitoring and unauthorized key exposure. Because client-side encryption and dynamic access control both focus on protecting health records, recent research has hypothesized that traditional centralized key management might naturally generalize to secure PHR sharing without needing expensive client-side re-encryption or pairing-based cryptography. This study tests that hypothesis using the SeSPHR framework. Within this framework, Setup and Re-encryption Server (SRS) and Web Crypto API architectures were evaluated across both portion-level and file-level sharing scenarios. Standard cryptographic metrics (encryption, re-encryption, and decryption latency) were used to evaluate the performance of the system. These operations were executed on various medical document sizes and tested on realistic healthcare workloads. Our results show that symmetric file encryption scaled solely on file sizes produces linear computational latency. On the other hand, the SRS proxy shows significantly greater execution consistency than local device re-encryption when exposed to changing viewer permissions when explicitly performing key re-wrapping. In the conclusion, this study shows that active key transformation and passive ciphertext storage are really separate concerns that require separate handling.**

***Index Terms*—Access control, Browser-side Cryptography, Personal health records, Proxy re-encryption.**

I. INTRODUCTION

The integration of cloud computing into digital healthcare systems has revolutionized modern medicine. By offering scalable storage, dynamic network accessibility, and collaborative data exchange, cloud platforms enable seamless coordination among patients, hospital clinics, diagnostic laboratories, medical researchers, and insurance providers. Among these advancements, Personal Health Records (PHRs) have emerged as a vital patient-centric tool. PHRs collect comprehensive patient medical data including demographics, diagnostic histories, laboratory results, pharmaceutical prescriptions, and private notes, providing a unified longitudinal view of a patient's health. Despite these clear operational benefits, storing highly confidential patient information on public, commercial cloud storage environments introduce severe security risks. Public cloud service providers are classically modeled as "honest-but-curious" entities. While they reliably execute storage and routing protocols, they remain highly motivated to analyze, mine, or leak patient medical records for advertising or research. Furthermore, centralized cloud databases are high-value targets for malicious external hackers and compromised administrative insiders. To satisfy legal compliance mandates, such as the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR), health data must remain completely secure and confidential. The only definitive protection against curious or compromised cloud operators is client-side cryptography, where records are encrypted on the patient's local device before they are uploaded. However, standard client-side encryption introduces a major operational bottleneck when sharing records with multiple readers or dynamic user departments. If a patient encrypts their health document using a single symmetric key and uploads it, sharing this document with a new physician requires either sharing the master key (compromising long-term security), or downloading the large ciphertext, decrypting it, and re-encrypting the entire file for the recipient. For low-power or mobile client devices, this process wastes

significant processor cycles and battery life, and demands high bandwidth. To solve this trilemma, this paper implements SeSPHR, a secure PHR sharing system based on policy-gated proxy re-encryption (PRE). Under SeSPHR, files are partitioned and encrypted on the client browser using a random 256-bit AES symmetric key in Galois/Counter Mode (GCM). The browser wraps (encrypts) this symmetric session key under the public key of a semi-trusted setup and re-encryption server (SRS) using RSA-OAEP. The cloud receives only the encrypted payload and the wrapped key blob. When an authorized physician requests access, the SRS evaluates policy conditions against the doctor's dynamic attributes. Upon validation, the SRS performs a key transformation routine, unwrapping and re-wrapping the key blob under the doctor's public key without revealing the underlying record. The doctor then decapsulates the key in their browser to decrypt the PHR. By transforming only, the 32-byte key material, SeSPHR ensures that key release operates at $O(1)$ constant-time.

The primary contributions of this paper are outlined as follows:

1. We design a hybrid cryptographic proxy re-encryption system that executes natively in modern browsers via standard Web Crypto APIs, avoiding custom plugins.
2. We implement portion-level access controls, allowing patients to separately encrypt and govern distinct sections of a single PHR document.

II. BACKGROUND

The crypto architecture of SeSPHR makes use of symmetric crypto algorithms, public key cryptosystem, and concepts of Proxy re-encryption to create an extremely secure and browser-compatible environment. Symmetric Crypto Algorithm: In symmetric key crypto system, the same single shared key is used for encryption as well as decryption. AES-GCM-256 algorithm makes use of the AES algorithm with a key size of 256 bits in Galois/Counter mode (GCM). GCM is an AEAD cipher, unlike older algorithms such as CBC. Asymmetric Key Wrap Algorithm: Asymmetric cryptography involves a pair of keys consisting of a public and a private key. RSA cryptography is mathematically prone to padding oracle attacks. Cryptographic Proxy Re-Encryption: Conventional proxy re-encryption schemes enable a proxy server, trusted only partially, to transform a ciphertext that is encrypted using the public key PK_A, to another one that is decryptable using the private key SK_B. Whereas conventional PRE mathematical schemes depend on pairing computations over some algebraic groups, the PRE technique in SeSPHR works differently.

III. LITERATURE SURVEY

Several privacy-preserving architectures have been explored in academic studies. Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [9] is widely cited as a robust solution for fine-grained access control. In CP-ABE, patient records are encrypted under descriptive Boolean policies, and users are issued private keys linked to personal attributes. If a user's attributes satisfy the policy, they can decrypt the record. Although CP-ABE provides elegant cryptographic policy enforcement, its practical implementation is severely limited. Decryption requires intensive bilinear pairings that lack native support in modern web browsers. Running ABE in JavaScript or WASM ports introduces extreme processing latency on mobile platforms. Furthermore, attribute revocation is notoriously difficult, requiring the data owner or central authority to re-encrypt all stored records and redistribute new keys to all non-revoked users. Traditional Proxy Re-Encryption models based on ElGamal or elliptic curves [4] allow a proxy to execute re-encryption keys, but they present significant key management overhead on the patient. The patient must remain online to calculate re-encryption parameters whenever a new reader joins, which is highly impractical for mobile users. Symmetric key access systems [8] use dynamic key-trees to manage group access. While computationally fast, these schemes suffer from key explosion at the client side. Any user revocation forces the patient to download the entire healthcare document, decrypt it, generate a new symmetric key, re-encrypt the file, and re-upload the massive ciphertext. This causes heavy bandwidth and CPU drain. Centralized Key Management Systems (KMS) [5] simplify key routing but introduce key escrow risks, as administrators hold master keys that can decrypt all stored medical records. SeSPHR resolves these architectural limits by combining browser-side symmetric file encryption

with an active SRS proxy. By performing RSA-OAEP key wrapping [7], SeSPHR ensures that private keys never leave the local browser, eliminating key escrow risks while keeping file sharing and revocation computationally lightweight. Furthermore, dynamic attribute-based evaluation in distributed systems has been proposed using blockchain or smart contracts [1] to eliminate centralized proxies. However, smart contract evaluations are public, exposing the patient's access policy and doctor attributes to the blockchain ledger. Centralizing policy evaluation inside the SRS [3] while keeping data strictly encrypted in the cloud strikes an optimal balance between security and performance.

IV. PROBLEM STATEMENT

Modern e-health sharing systems struggle to balance three competing objectives: absolute patient data privacy, fast execution speeds on web browsers, and low-cost user access revocation. Plaintext cloud storage exposes patient records to hacking and unauthorized mining. Conversely, standard client-side encryption introduces high latency on client devices. Whenever access is granted or revoked, the patient must download, decrypt, re-encrypt, and re-upload multi-megabyte files, consuming severe processing power and network bandwidth. Centralized key distribution services solve these latency issues but introduce single points of failure and key escrow vulnerabilities, where compromised administrators can access decrypted files. Furthermore, advanced cryptographic schemes like CP-ABE lack native support in standard browsers, leading to unacceptable delays. Therefore, the problem is to design and implement a secure PHR sharing system that guarantees patient-centric data ownership, runs natively in standard web browsers without plugins, isolates the proxy and cloud from file plaintext, and achieves constant-time key transformation and instant user revocation without requiring file re-encryption.

V. PROPOSED SOLUTION

To resolve the security and efficiency trilemma, we propose SeSPHR, a framework that isolates the secure storage of file ciphers in the cloud while delegating access validation and key transformations to a semi-trusted SRS proxy. SeSPHR separates the data plane from the control plane. Patients perform client-side file encryption using a unique, random symmetric AES key. This AES key is then wrapped under the SRS's public key, preventing the cloud or external attackers from reading the file key. When a request accessed by a physician, the SRS will evaluate the policy conditions against the doctor's dynamic attributes which is stored in a secure SQLite database. If authorized, the SRS executes a secure unwrapping and re-wrapping routine in shielded encrypted memory. It decrypts the wrapped key using its private key and immediately encrypts it using the doctor's public key. The SRS only processes the 32-byte key material, keeping the key-release phase highly efficient and constant-time $O(1)$ regardless of file size. Access revocation is accomplished instantly on the SRS by adding the doctor's ID to a revoked list or changing the file policy, halting future key transformations without modifying the stored ciphertext.

VI. METHODOLOGY

The cryptographic protocol sequence of the SeSPHR framework comprises the following operational phases: System Initialization & User Registration: The SRS generates its root keypair. During registration, each user generates an RSA-2048 keypair natively in their browser. The public key is uploaded to the SRS database, while the private key is retained securely in browser storage. Client-Side File Encryption (Patient): When a patient uploads a PHR file, the browser client generates a random 256-bit symmetric session key and a random 12-byte initialization vector. The symmetric key is then encapsulated (wrapped) under the SRS public key using RSA-OAEP with SHA-256.The patient uploads the ciphertext, wrapped key blob to the cloud. Key Transformation & Re-Encryption (SRS): When a doctor requests access to the file, the SRS retrieves the doctor's dynamic attributes and verifies them against the file access policy. If policy checks succeed, the SRS performs key translation. The SRS immediately encrypts (re-wraps) the recovered under the doctor's public key using RSA-OAEP with SHA-1. The transformed key blob is sent back to the doctor. Client-Side Decryption & Record Recovery: The doctor's browser downloads the ciphertext and the re-wrapped key blob. It unwraps using its private

key. Finally, the browser decrypts the ciphertext using and to recover the plaintext record

Algorithm I: Client-Side Portioned Encryption & Metadata Generation

Input: PHR Document partitions M_i, Access Policies P_i, SRS Public Key PK_{SRS}

Output: Encrypted files C_i, Metadata JSON containing key blobs $W_{SRS,i}$

1. Generate user asymmetric keys: $SK_u, PK_u \leftarrow$ crypto. subtle.generateKeyPair()
2. for each partition M_i do
3. Generate a random 256-bit symmetric session key $K_{AES,i}$ and initialization vector IV_i
4. Compute authenticated ciphertext: $C_i \leftarrow$ AES-GCM-Encrypt$(K_{AES,i}, M_i, IV_i)$
5. Encapsulate key: $W_{SRS,i} \leftarrow$ RSA-OAEP-Wrap$(PK_{SRS}, K_{AES,i})$
6. end for
7. Construct structural metadata block:
8. Metadata ← {owner_uuid,portions = $[(name_i, P_i, W_{SRS,i}, IV_i)]$}
9. Transmit {$C_i$andMetadata to Cloud Storage

Algorithm II: SRS Policy-Gated Proxy Re-Encryption

Input: Access Request from Doctor D for file F, dynamic Attribute Set $Attr_D$, Doctor Public Key

Output: Transformed key blobs $W_{D,i}$ for authorized record portions

1. Query database: $Attr_D \leftarrow$ GetAttributes(D.user_id)
2. if Evaluate Policy($Attr_D$, F.global_policy) = False then
3. Log denial and return Reject Access
4. end if
5. if D.user_id ∈ F.revoked_users then
6. Log denial and return Reject Access
7. end if
8. Initialize accessible portions ←
9. for each portion P_i inside F.portions do
10. if Evaluate Policy($Attr_D$, P_i.policy) = True then
11. Decapsulate file key: $K_{AES,i} \leftarrow$ RSA-OAEP-Unwrap$(SK_{SRS}, P_i.W_{SRS,i})$
12. $(portion\ name_i, W_{D,i}, P_i.IV_i)$toaccessible_portions
13. end if
14. end for
15. Log ACCESS event in SHA-256 tamper-evident log chain
16. return accessible portions to doctor browser client

VII. ARCHITECTURE

The logical design of SeSPHR isolates the active control plane (key management and policy evaluation at the SRS) from the passive data plane (untrusted cloud storage).

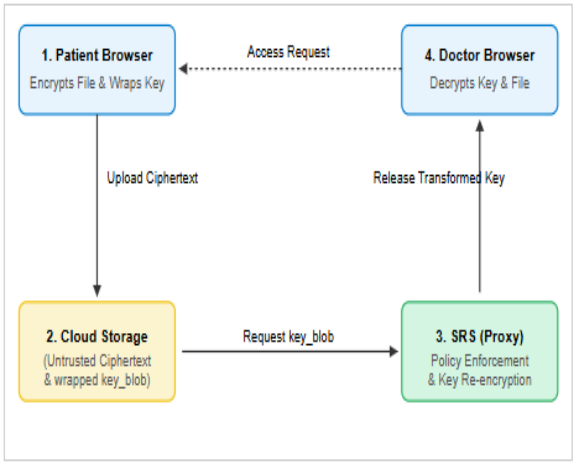

Figure 1 Logical architectural layout of SeSPHR system, isolating passive data streams from active policy-gated key transformation operations.

An essential innovation in the SeSPHR architecture is Fine-Grained Portion-Level Access Control. Instead of treating the entire medical file as a single block governed by a single policy, SeSPHR logically partitions a PHR document into separate functional sections:

1. Personal Information: Patient's name, phone, and address.
2. Medical History: Allergies, previous operations, and diagnosis.
3. Insurance Information: Billing records, policies, and claims.
4. Pharmaceutical Prescriptions: Daily dosage and medication lists.

Each portion is encrypted using a distinct symmetric session key, and wrapped separately under the SRS key. This allows patients to customize access per portion. For example, a pharmacist is granted access only to the pharmaceutical portion, an insurance representative is restricted to the billing portion.

VIII. IMPLEMENTATION

The SeSPHR system has been fully implemented. The technological stack consists of: Backend Stack (Flask & SQLite): The API server is developed in Python using the Flask framework. The backend manages the registration of medical users and handles the evaluation of access policies. The SQLite database engine incorporates active foreign keys, mapping user profiles to their attributes. The relational tables used are detailed in the following schema block:

```
-- SQL Schema for User and Attributes Management
CREATE TABLE IF NOT EXISTS users (
    user_id TEXT PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    name TEXT,
    password_hash TEXT NOT NULL,
    role TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS attributes (
    user_id TEXT,
    key TEXT NOT NULL,
    value TEXT NOT NULL,
    FOREIGN KEY(user_id) REFERENCES users(user_id) ON
DELETE CASCADE,
    PRIMARY KEY (user_id, key)
);
```

Figure 2 SQL Schema for User and Attributes Management

Cryptographic Core: Cryptographic routines on the server utilize PyCryptodome. The SRS root keypair is stored in PKCS format on disk. Browser-side cryptography utilizes the native Key generation is handled via:

```
window.crypto.subtle.generateKey(
    { name: "AES-GCM", length: 256 },
    true,
    ["encrypt", "decrypt"]
);
```

Figure 3 Cryptographic Function Logic

Blockchain-Style Auditing: Every system action including access requests, key transformations, dynamic updates, and revocations is logged sequentially using a blockchain-like hash chain. When an event is recorded, the SRS locks the log file to prevent parallel write forks, reads the previous entry's SHA-256 hash H_{i-1}, and generates the new entry's cryptographic signature: H_i=SHA256(Entry$_i$UH_{i-1}) The admin interface features an automated chain-validation engine. Any manual modification or entry injection immediately breaks the hash chain, exposing unauthorized database editing or administrative tampering.

IX. RESULTS AND DISCUSSION

To measure the performance and scalability of SeSPHR, we executed administrative benchmarks across multiple record scales, ranging from 100 KB to 10 MB.

Table 1 Empirical Cryptographic Timings

File Size	Encryption (s)	SRS Re-enc (s)	Decryption (s)
100 KB	0.01187	0.20444	0.06726
1 MB	0.00581	0.19800	0.07735
5 MB	0.01220	0.18054	0.08157
10 MB	0.02647	0.18338	0.09548

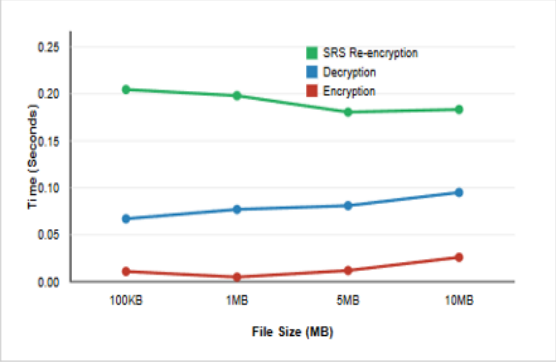

Figure 4 The constant-time complexity of the SRS key.

The constant-time response of the key transformation represents a major advancement for secure medical storage frameworks. Since the SRS only translates the 32-byte session key rather than processing the multi-megabyte record payload, key release does not exhaust CPU or network resources. Storing the file ciphers in the cloud while retaining wrapped keys ensures that the cloud has zero readability, while the tamper-evident audit logs preserve operational transparency. Because the cloud cannot decrypt key blobs, collusion between a malicious cloud operator and a revoked doctor fails to expose patient files. This satisfies forward and backward security properties. Threat Model & Security Discussion: SeSPHR operates under a

robust threat model, defending against multiple distinct attack vectors:

1. Snooping Cloud Provider: The cloud stores only AES ciphertexts and RSA wrapped keys. Because standard RSA is semantically secure under OAEP, the cloud learns nothing about the files or the symmetric keys.
2. Malicious SQL/DB Administrator: An insider attempting to bypass policy evaluation by directly reading or writing to the database cannot retrieve the files. Since medical ciphers reside strictly in the cloud.
3. Collusion Attacks: Collusion between a compromised cloud service provider and a revoked physician fails. The cloud possesses no decryption capabilities, and the SRS rejects key translation for the revoked user.

X. CONCLUSION

In this paper, we presented SeSPHR, a browser-side encrypted personal health record sharing framework with policy-gated proxy re-encryption. By combining client-side AES-256-GCM symmetric file encryption with asymmetric RSA-2048 key wrapping at a semi-trusted SRS proxy, the scheme secures patient health records against curious cloud providers. Experimental benchmarks validate that the key re-encryption phase operates at $O(1)$ constant-time (~0.18s), making the framework highly scalable. Operational transparency is guaranteed through cryptographically chained, tamper-evident SHA-256 audit logs. Future work will focus on integrating zero-knowledge proofs (ZKPs).

REFERENCES

[1] M. Ali, A. Abbas, M. U. S. Khan, and S. U. Khan, "SeSPHR: A Methodology for Secure Sharing of Personal Health Records in the Cloud," IEEE Transactions on Cloud Computing, vol. 6, no. 4, pp. 1102–1115, Oct.–Dec. 2018.

[2] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, pp. 1–30, Feb. 2006.

[3] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Transactions on Information Theory, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

[4] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in Proc. IEEE INFOCOM, San Diego, CA, USA, 2010, pp. 1–9.

[5] M. Jafari, R. S. Naini, and N. P. Sheppard, "A rights management approach to protection of privacy in a cloud of electronic health records," in Proc. 11th ACM Workshop on Digital Rights Management (DRM), Chicago, IL, USA, 2011, pp. 23–30.

[6] J. Xu, B. Zhang, and X. Wang, "A Blockchain-Based Secure Sharing Scheme for Electronic Health Records Using Proxy Re-Encryption," Journal of Medical Systems, vol. 44, no. 5, p. 94, 2020.

[7] X. Liang, Z. Cao, H. Lin, and J. Shao, "Attribute-based proxy re-encryption with delegating capabilities," in Proc. 4th ACM Symposium on Information, Computer and Communications Security (ASIACCS), Sydney, Australia, 2009, pp. 276–286.

[8] R. Jiang and Y. Wang, "Proxy Re-Encryption with Cryptographic Auditing and Forward/Backward Revocation Control for E-Health Data Sharing," Information Sciences, vol. 542, pp. 312–329, 2020.

[9] S. D. S. Al-Shehri, "Fine-Grained Sharing of Health Records in Cloud Using Portion-Level Encryption," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2435–2448, 2020.

[10] R. Jiang and Y. Wang, "Proxy Re-Encryption with Cryptographic Auditing and Forward/Backward Revocation Control for E-Health Data Sharing," Information Sciences, vol. 542, pp. 312–329, 2020.