



An Adaptive and Risk-Aware Access Control Framework for Secure Cloud File Sharing Using Hybrid Cryptography and Context-Aware Key Release

Ghodekar Snehal Ramesh¹, Ghodekar Vaishnavi Suresh², Miss. Ghodake G. K.³

Student, Department of MCA Engineering, Samarth College of Engineering and Management, Belhe, Pune, India¹

Student, Department of MCA Engineering, Samarth College of Engineering and Management, Belhe, Pune, India²

Guide, Department of MCA Engineering, Samarth College of Engineering and Management, Belhe, Pune, India³

Abstract: Cloud platforms have become a common place to keep business records because they are easy to use and accessible from anywhere. Still, sensitive documents stored this way face two problems: the service provider should not be able to read them, and a password alone should not decide whether a request is genuine. This study proposes ARAC-HC, a framework where the key needed to open a document is released only after the system judges the situation of each request. At upload time, the document is protected on the user's own device with a 256-bit AES-GCM key. This key is then wrapped separately for every approved reader through a short-lived X25519 exchange followed by HKDF. Before handing over the wrapped key, the framework checks seven signals: device state, location, access time, failed login attempts, recent user activity, stored trust level and the sensitivity of the requested document. Based on the calculated risk, the system either allows normal access, asks for an extra check, permits restricted access or refuses the request. In tests, protecting a 100 MB document took close to 75 ms. The X25519-based method finished key protection roughly 2.6 times faster than RSA-2048. Across 20,000 simulated requests, the framework caught 95.3% of malicious attempts, and 97% of legitimate users got access without any extra verification step.

Keywords: Context-aware key release, risk-based authorisation, cloud document sharing, envelope encryption, AES-GCM, X25519, insider misuse, zero trust.

I. INTRODUCTION

Companies today place many important records on cloud platforms, including tender papers, payroll details and program source code. Workers may open these records from home networks, hotels or personal phones that also run several other apps. Industry surveys flag account takeover and insider misuse as two of the biggest cloud-security worries [1], [2].

Most existing systems pair a permission list with encryption. Role-based or attribute-based methods mainly check whether an account may open a given document. They seldom look at what the user is doing right now. So, when an attacker gets a valid password and signs in from an unknown device at a strange hour, the platform may treat the request as normal. Attribute-based encryption puts the access rule inside encrypted data, yet it needs costly pairing operations and makes revoking access hard [6]-[10].

Zero-trust thinking says every request should be judged using present conditions instead of an earlier successful login. In document sharing, a risk score helps only when it directly decides whether the decryption key is released. If an attacker can skip the application layer and read stored data, a high risk score by itself cannot stop leakage.

ARAC-HC keeps document protection and key distribution separate. First, the document is encrypted using AES-256-GCM, which gives confidentiality and detects tampering [18], [19]. Next, the document key is wrapped one by one for each approved reader through a one-time X25519 exchange and HKDF [20], [21]. The gatekeeper releases this wrapped key only when the risk engine allows the request. Thus, denial is enforced by holding back the key rather than by an application-level rule alone.

The paper contributes: an architecture where risk decisions directly control encrypted-key delivery; a light scoring model using five live context signals plus trust and file sensitivity; a low-cost revocation method; and a prototype tested with timing runs and 20,000 simulated requests.



II. LITERATURE REVIEW

A. Encryption-Based Policy Enforcement

Sahai and Waters first suggested using attributes as a condition for decryption [6]. Later work produced key-policy and ciphertext-policy attribute-based encryption. In ciphertext-policy ABE, the document owner decides who may decrypt the content [8]. Yu et al. joined key-policy ABE with proxy re-encryption to cut revocation costs [9]. Wan et al. built a hierarchy of key-issuing authorities [10], and Li et al. applied several authorities for medical records [11]. Proxy re-encryption lets a partly trusted server convert ciphertext for another user without seeing the original document [12], [13]. These methods give strong policy control, but they often need heavy computation, use fixed policies and pay little attention to live user context.

B. Risk-Driven Authorisation

Cheng et al. introduced the Fuzzy MLS model, where risk comes from the gap between user clearance and object classification [14]. Kandala et al. treated risk and trust as attributes in an authorisation framework [15]. Shaikh et al. estimated risk from past user behaviour [16], while Atlam et al. combined device data with user activity for IoT settings [17]. These studies show that context improves access decisions. However, the decision is usually enforced only by an application component. If that component is bypassed, stored data may still be exposed.

C. Research Gap

Encryption-based methods protect data from the cloud provider but do not react to changing user context. Risk-based methods react to context but often leave stored documents readable by the storage provider. ARAC-HC merges both ideas by encrypting documents on the client and using risk-aware decisions to control key release.

III. SYSTEM MODEL

A. Participants

The framework has an uploader, one or more readers, local agents, a gatekeeper and a storage host. The uploader picks approved readers and assigns a sensitivity level to each document. Every user device runs a local agent for encryption, decryption and context collection. Private keys stay only on user devices. The gatekeeper maintains the risk-evaluation module and trust-related records, and it determines whether an encrypted key should be provided to the requesting user. It never gets a document key in plain form. The storage host keeps encrypted documents and wrapped keys but cannot read their contents.

B. Adversary Assumptions

The cloud storage provider is treated as semi-honest. It follows the protocol but may try to learn document contents, and its storage may be leaked. External attackers may use stolen passwords, replay captured messages or run automated login attempts. Insiders may use valid accounts and trusted devices but may download unusually large amounts of data. Physical attacks on user devices and side-channel attacks on cryptographic libraries are outside this study [23].

C. Objectives

ARAC-HC aims to keep documents confidential and tamper-evident, spot compromised accounts and insider misuse, and apply a response that matches the calculated risk so normal users are not disturbed needlessly.

IV. METHODOLOGY

A. System Architecture

The framework has three layers. The client layer holds uploader and reader agents. The control layer contains cryptographic services, context collection, the gatekeeper and a hash-linked trust ledger. The cloud layer stores encrypted documents and wrapped keys. Plain document content and plain document keys appear only briefly inside a user agent.

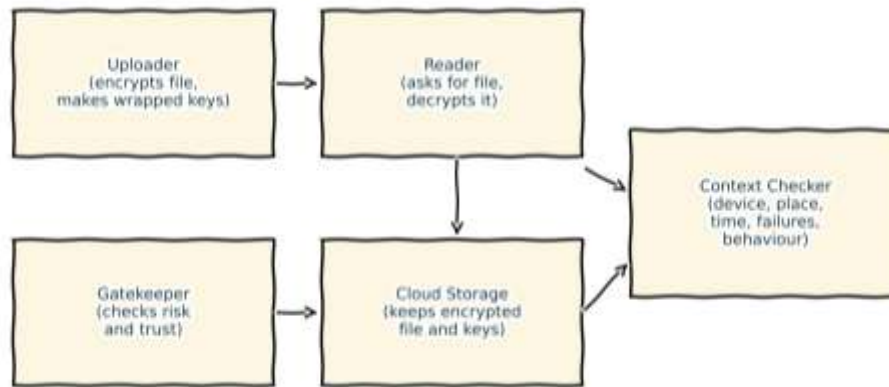


Fig. 1 Working structure of the proposed ARAC-HC system

B. Hybrid Cryptographic Scheme

The framework uses envelope encryption. A symmetric key protects the document, while asymmetric cryptography protects that symmetric key for each recipient.

When a document F is uploaded, the agent creates a random 256-bit key k and a 96-bit nonce N . The protected document is obtained as $C = \text{AES-GCM.Enc}(k, N, F, \text{AD})$. Here, AD includes the document identifier, owner identifier and hash of the access policy. This associated data stops replacement of ciphertext or change of the policy.

For every reader u , the uploader agent makes a temporary X25519 key pair. It computes a shared secret with the reader public key and derives a key-encryption key using HKDF-SHA256. Using the derived key-encryption key, the system encrypts the document key k , thereby generating the wrapped key W_u . The temporary private key is deleted right away. Since the wrapped key is tied to the document identifier and reader public key, it cannot be reused for another document or user.

During access, an approved reader gets W_u , computes the same shared secret using its private key, recovers k and decrypts the document. The gatekeeper only forwards the wrapped key and cannot decrypt any document because it never has the reader private key.

C. Risk Score Calculation

The context collector gives five normalised signals: device risk, location risk, time risk, failed-login risk and behavioural risk. These signals are combined using assigned weights. Device and location risks spot unfamiliar access environments, failed-login risk catches automated attempts, and behavioural risk spots unusual download or file-access activity. The context score is then joined with user trust and document sensitivity to get the final risk value R .

D. Response Levels

If R is below 0.35, the wrapped key is released normally. If R is between 0.35 and 0.50, the user must finish OTP or FIDO2 verification before access. If R is between 0.50 and 0.65, access is given in read-only mode after verification. If R is 0.65 or above, the wrapped key is withheld, the uploader is told and re-keying may be scheduled.

E. Trust Update and Revocation

After every session, user trust is updated using a decay factor. Successful sessions raise trust slowly, while blocked or suspicious sessions lower it. When access must be cancelled, the wrapped key of that reader is removed at once. A new document key can be made during the next document update and wrapped only for remaining authorised readers. This avoids changing keys for all users.

V. SECURITY ANALYSIS

The storage host holds only encrypted documents and wrapped keys. Without the reader private key or the deleted temporary key, the document key cannot be recovered. So, leakage of cloud storage does not expose document contents. AES-GCM authentication covers both encrypted data and associated data. Any change in document bytes, ciphertext identity or policy information makes decryption fail.

A stolen password alone does not give access because the reader private key stays on the user device. If an attacker also gets the device, unusual device, location and failed-login signals raise the risk score and demand extra verification. Insider misuse is caught mainly through behaviour analysis. Large downloads or access to unfamiliar documents raise behavioural risk and lower trust. The restricted level stops saving until the activity is reviewed.



Fresh nonces and short-lived signed tokens stop replay of old key deliveries.

VI. RESULTS AND DISCUSSION

A. Test Environment and Cryptographic Cost

The prototype was built in Python using cryptographic libraries for AES-GCM, HKDF, X25519, RSA-OAEP and ECDSA. Experiments ran on a virtual machine with two virtual CPUs. Reported values are medians from repeated runs. X25519 key generation took 0.034 ms, while RSA-2048 and RSA-3072 took 62.63 ms and 163.58 ms respectively. A full X25519-based key wrapping round took 0.126 ms, against 0.325 ms for RSA-2048 and 0.890 ms for RSA-3072. The wrapped key size was only 92 bytes, cutting storage needs by 64% to 76% versus RSA-based methods.

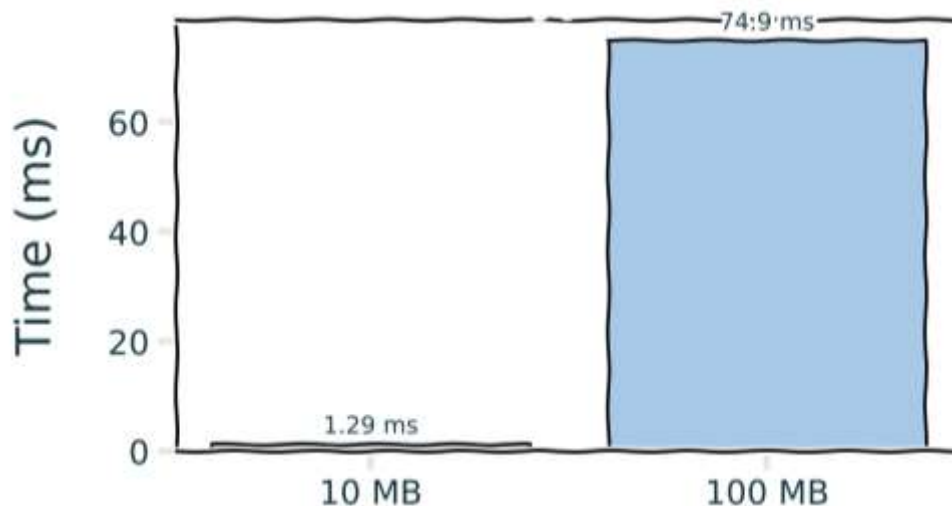


Fig. 2 AES-GCM encryption time for 10 MB and 100 MB files

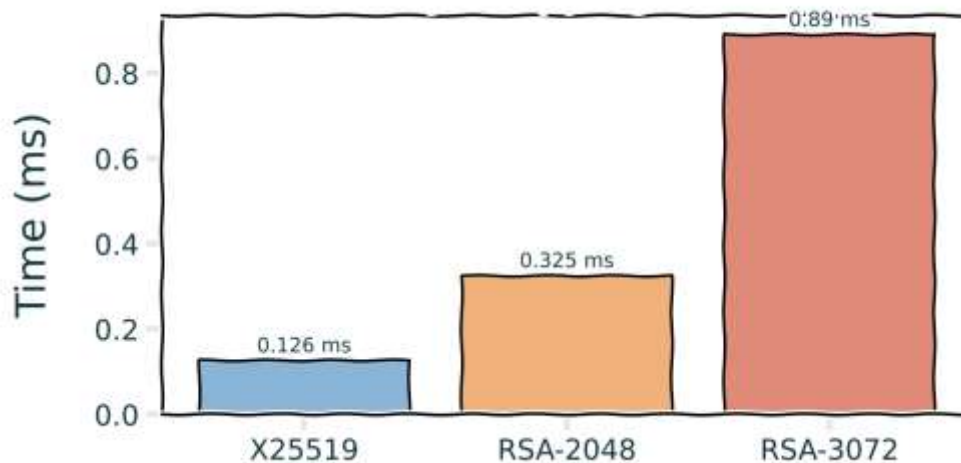


Fig. 3 Key wrapping time comparison of X25519, RSA-2048 and RSA-3072

Protecting a 10 MB document took 1.29 ms, while a 100 MB document took 74.9 ms. The encryption speed was near 1.30 GB/s due to AES-NI support. For 200 readers, wrapping all keys took 16.53 ms, showing the method scales linearly and stays practical.

B. Attack Detection

For attack detection, 20,000 simulated requests were made, with 85% benign and 15% malicious. Malicious cases included hijacked accounts, insider misuse and automated bot activity. ARAC-HC got 94.2% accuracy, 0.738 precision,



0.953 recall and 0.832 F1-score. It caught 95.3% of malicious requests, against 9.6% for a role-based check and 62.5% for attribute-based rules. Among genuine users with valid roles, 97% got direct access without extra verification.

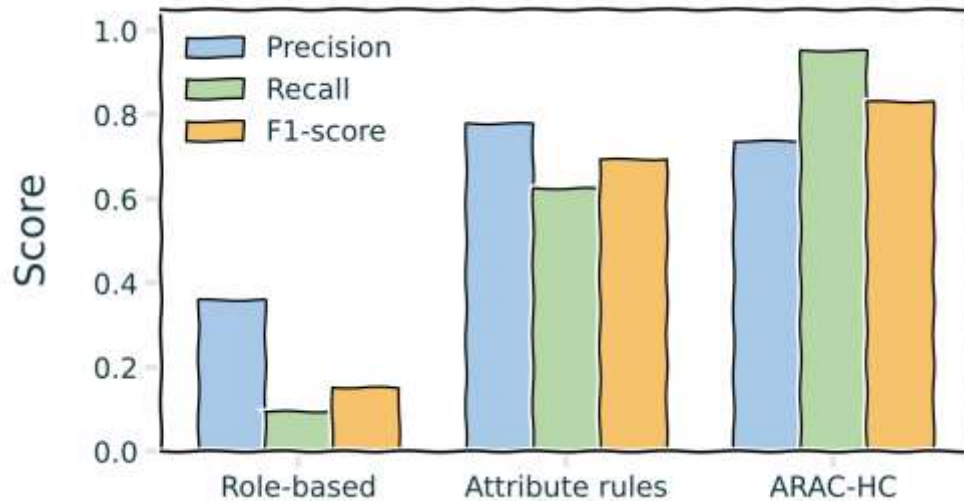


Fig. 4 Detection performance of the compared access-control methods

The proposed method caught 99.1% of hijacked-account requests and all bot requests. For insider misuse, behaviour and trust info enabled detection of 87% of cases, while attribute rules caught only 8.8%. The threshold value 0.35 gave a strong balance between attack detection and user convenience.

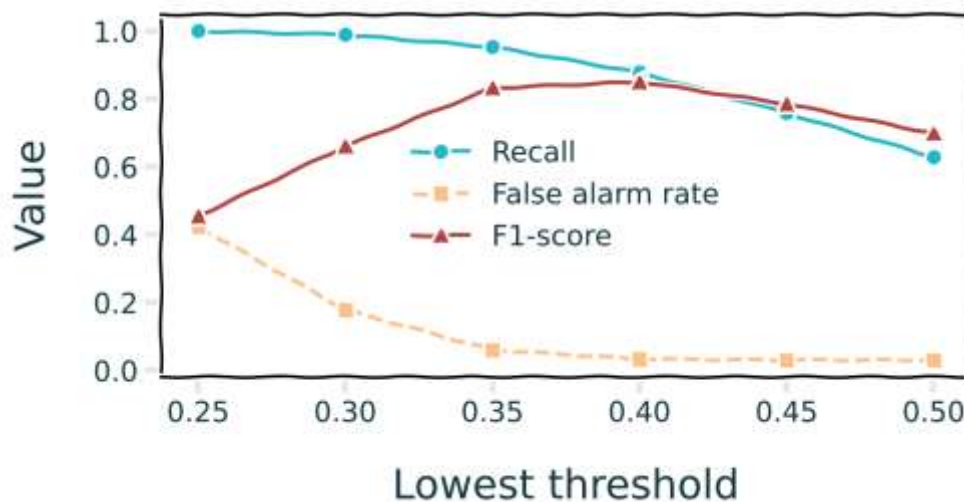


Fig. 5 Effect of the lowest risk threshold on recall, false alarm rate and F1-score

VII. CONCLUSION

This paper introduced ARAC-HC, a cloud document-sharing framework that joins client-side encryption with context-aware risk-based access control. Documents are protected using AES-256-GCM, and document keys are wrapped one by one for each reader using X25519 and HKDF. The gatekeeper releases a wrapped key only after checking device, location, time, login failures, behaviour, trust and document sensitivity.

Experimental results show the framework gives fast cryptographic operations and improves attack detection versus role-based and attribute-based methods. In future work, model weights can be learned from real anonymised access logs, thresholds can match organisational risk policies, and post-quantum key encapsulation can be added for long-term protection.

**VIII. ACKNOWLEDGMENT**

The authors thank the Department of MCA Engineering, Samarth College of Engineering and Management, Belhe, for the required support and computing facilities.

REFERENCES

- [1]. P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, 2011.
- [2]. Cloud Security Alliance, "Top threats to cloud computing," CSA Research Report, 2022.
- [3]. R. S. Sandhu et al., "Role-based access control models," IEEE Computer, vol. 29, no. 2, pp. 38-47, 1996.
- [4]. V. C. Hu et al., "Guide to attribute based access control," NIST Special Publication 800-162, 2014.
- [5]. S. Rose et al., "Zero trust architecture," NIST Special Publication 800-207, 2020.
- [6]. A. Sahai and B. Waters, "Fuzzy identity-based encryption," Proc. EUROCRYPT, LNCS 3494, 2005, pp. 457-473.
- [7]. V. Goyal et al., "Attribute-based encryption for fine-grained access control," Proc. ACM CCS, 2006, pp. 89-98.
- [8]. J. Bethencourt et al., "Ciphertext-policy attribute-based encryption," Proc. IEEE S&P, 2007.
- [9]. S. Yu et al., "Achieving secure, scalable, and fine-grained data access control in cloud computing," Proc. IEEE INFOCOM, 2010.
- [10]. Z. Wan et al., "HASBE: A hierarchical attribute-based solution," IEEE Trans. Information Forensics and Security, vol. 7, no. 2, 2012.
- [11]. M. Li et al., "Scalable and secure sharing of personal health records," IEEE Trans. Parallel and Distributed Systems, vol. 24, no. 1, 2013.
- [12]. M. Blaze et al., "Divertible protocols and atomic proxy cryptography," Proc. EUROCRYPT, 1998.
- [13]. G. Ateniese et al., "Improved proxy re-encryption schemes," ACM Trans. Information and System Security, vol. 9, no. 3, 2006.
- [14]. P.-C. Cheng et al., "Fuzzy multi-level security," Proc. IEEE S&P, 2007, pp. 222-230.
- [15]. S. Kandala et al., "An attribute based framework for risk-adaptive access control," Proc. ARES, 2011.
- [16]. R. A. Shaikh et al., "Dynamic risk-based decision methods," Computers & Security, vol. 31, no. 4, 2012.
- [17]. H. F. Atlam et al., "Developing an adaptive risk-based access control model for IoT," Proc. IEEE iThings, 2017.
- [18]. NIST, "Advanced Encryption Standard (AES)," FIPS 197, 2001.
- [19]. M. Dworkin, "Recommendation for block cipher modes: GCM and GMAC," NIST SP 800-38D, 2007.
- [20]. D. J. Bernstein, "Curve25519," Proc. PKC, LNCS 3958, 2006.
- [21]. H. Krawczyk, "Cryptographic extraction and key derivation: HKDF," Proc. CRYPTO, 2010.
- [22]. R. L. Rivest et al., "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, 1978.
- [23]. T. Ristenpart et al., "Hey, you, get off of my cloud," Proc. ACM CCS, 2009.