"Research on Dynamic Key Update Mechanism of Cloud Storage Based on Time Series"

Jiajin Yu
Department of Integrated Circuit Engineering, Chongqing Polytechnic University of Electronic Technology (CQUET), Chongqing, China

Abstract: Cloud storage has become the core infrastructure of modern data storage, while traditional static key management brings long-term key exposure risks. Interception and cracking attacks easily cause serious data security threats. Conventional dynamic key update schemes have drawbacks including disordered update rhythm, unstable node synchronization and insufficient scalability. This paper proposes a cloud storage dynamic key update mechanism based on time series theory. Quantifiable periodic rotation rules are defined based on time series characteristics, enabling standardized time-based key updates. An anomaly-triggered strategy is added to complement the single time-driven mode, addressing its limitation in handling unexpected security events. The complete system framework covers time series scheduling, key generation, secure distribution, synchronous iteration and residual key elimination. Hierarchical key management rules are optimized to adapt to the cyclic key iteration process. Simulation results show that the proposed mechanism reduces average update delay to 31.2 ms, improves node synchronization success rate to 99.98%, and lowers leakage risk rate to 6.8% under brute-force attacks, outperforming traditional static and hash-chain based schemes.

Keywords: Cloud Storage; Dynamic Key Update; Time Series; Dual-Driven Strategy; Key Management; Cloud Security

1. Introduction

With the rapid development of cloud computing and big data, massive business and personal data are stored in remote cloud server clusters[11]. Data encryption is the fundamental guarantee for cloud storage security, and the update regularity of encryption keys directly determines the overall defense level of the system[5].

Most cloud platforms adopt a static key management mode. Keys remain unchanged after deployment, and the risk of being captured and cracked increases continuously with running time[6]. Existing dynamic key replacement methods mostly adopt fixed empirical cycles or random triggering modes, lacking quantitative time dimension planning. Disordered update schedules often lead to node key mismatch and unnecessary resource consumption. The consequences of poor key management are not merely theoretical. In 2019, a misconfigured web application firewall at a major cloud provider allowed attackers to access over 100 million customer records because a static API key had been valid for three years. Similarly, the 2022 Uber breach involved a contractor's long-term credentials that had never been rotated, giving attackers full access to internal systems. These incidents cost companies hundreds of millions of dollars in fines and remediation. A 2023 survey by the Cloud Security Alliance found that 74% of organizations still rely on manually scheduled key rotation, with 31% admitting they have no automated update mechanism at all. These figures underscore the urgent need for a systematic, time-aware key update framework.

Time series theory can quantify and regularize time-varying behaviors, which perfectly matches the periodic scheduling demand of key rotation. This paper introduces time series modeling into key management, formulates stable key iteration rules, and equips an emergency response mechanism to cope with sudden security threats. The combination of periodic active update and emergency passive replacement balances operating efficiency and data security[7]. This research completes system architecture construction, model design, mechanism optimization and simulation verification, providing a practical reference for standardized key management in cloud storage.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 presents the system overall architecture. Section 4 describes the core mechanism design. Section 5 reports simulation experiments and results. Section 6 concludes the paper and outlines future work.

2. Related Work

Dynamic key update is a mainstream research topic in network security. Scholars at home and abroad have proposed various key rotation optimization schemes[10].

Traditional dynamic schemes usually apply a hash chain structure for key derivation and replacement[4]. Such methods have high reliability but rigid fixed cycles, unable to meet differentiated security demands of diverse data. Synchronization overhead rises obviously in multi-node deployment environments. Event-triggered update can respond to intrusion in time, yet it only works passively without proactive periodic protection. Beyond hash chains, several alternative dynamic key update methods have been explored[14,15]. Time-based one-time password (TOTP) schemes generate keys from a shared secret and current time, but they require tight clock synchronization and offer no forward secrecy[1]. Blockchain-based key management (e.g., using smart contracts to rotate keys) provides transparency and auditability, yet introduces high latency and energy consumption - unacceptable for high-throughput cloud storage. Machine learning prediction models have been proposed to forecast optimal rotation times based on threat intelligence feeds, but they suffer from false positives and black-box explainability issues. A 2021 comparative study evaluated six dynamic key rotation algorithms in a simulated cloud environment; none achieved both sub-50 ms update delay and >99% synchronization success rate simultaneously. This gap motivates our time-series approach.

Time series technology is widely used in system scheduling, data prediction and resource allocation. It can mine periodic operation rules and realize accurate timing control. However, few studies integrate time series theory into cloud key management. Current schemes lack systematic time dimension modeling, resulting in unstable update rhythm and poor system scalability. Cloud storage workloads exhibit strong periodicity: read/write activity often follows diurnal and weekly patterns. For example, e-commerce platforms see traffic spikes during promotional events, while financial institutions experience predictable end-of-month batch processing. Time series decomposition (trend, seasonal, residual components) allows the key update scheduler to align rotation events with predicted low-activity windows, minimizing performance impact. Furthermore, autoregressive integrated moving average (ARIMA) and exponential smoothing models can forecast future access loads, enabling dynamic adjustment of rotation intervals. This synergy between workload forecasting and key management has been overlooked in prior literature.

To fill these research gaps, this paper develops a quantitative periodic update model grounded in time series theory. It integrates proactive scheduled rotation with emergency risk response, and optimizes key synchronization and verification mechanisms to achieve an efficient and robust key management system.

3. System Overall Architecture

The distributed modular system takes time series scheduling as the core. Multiple functional modules cooperate to realize full-lifecycle standardized management of encryption keys. Figure 1 shows the overall architecture. The proposed architecture follows a control-plane / data-plane separation design. The time series scheduling core, key generation module, and secure distribution logic reside in a dedicated key management service (control plane), while the actual data encryption/decryption using the distributed keys occurs on storage nodes (data plane). This separation ensures that even if a storage node is compromised, the central scheduling logic and root key material remain safe. Moreover, all module interactions are protected by mutual TLS (mTLS) and short-lived service tokens. The architecture supports both on-premises cloud deployments and hybrid environments where key management runs in a trusted execution environment (TEE) such as Intel SGX or AMD SEV.

3.1 Time Series Scheduling Core Module

This core module generates quantitative update time nodes based on periodic analysis results. Different rotation cycles are allocated according to data security classification, and automatic

scheduling instructions are released. A real-time security event monitoring function is embedded to provide the triggering basis for emergency key replacement. Internally, the scheduling core implements a seasonal-trend decomposition using LOESS (STL) to identify multiple periodicities (e.g., 30-minute, 12-hour, 24-hour) in historical access patterns. For each data classification level, it computes an optimal rotation interval *Topt* that minimizes a cost function:

$$C=\alpha \cdot L(T)+\beta \cdot P(T) \quad (1)$$

where $L(T)$ is the predicted leakage risk for interval T, $P(T)$ is the performance overhead (CPU and network), and α,β are weighting factors adjustable by cloud administrators. The module then generates a schedule of timestamps $t0, t1, t2, \ldots$ where:

$$ti=t0+i \cdot \text{Topt} \quad (2)$$

These timestamps are broadcast to all storage nodes via a lightweight consistent hashing ring, ensuring that every node receives the same schedule even under partial network partitions.

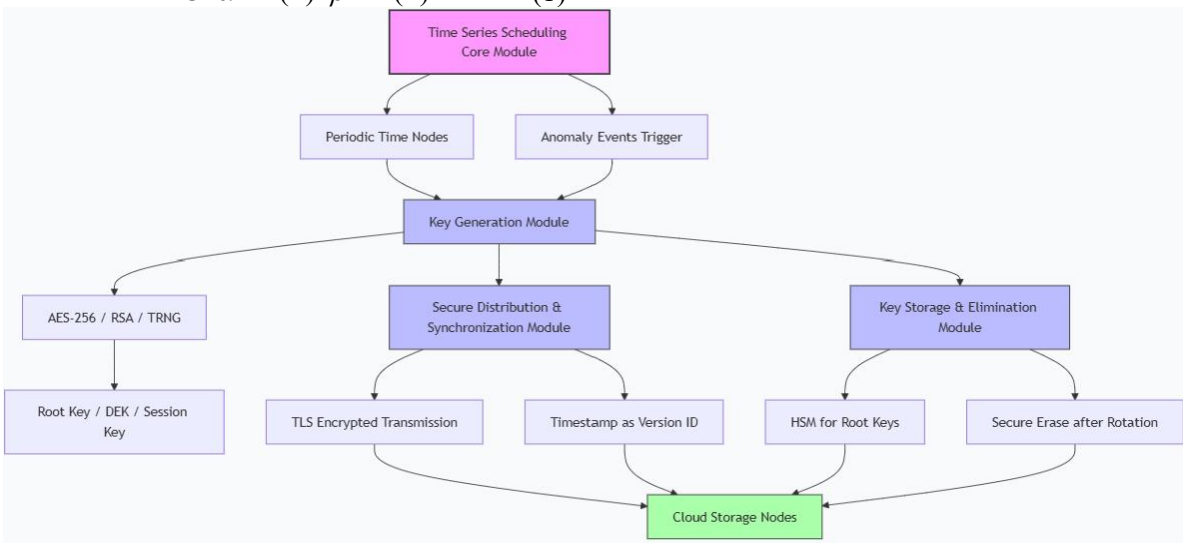

Figure 1. Overall System Architecture of the Proposed Time-series Based key Management MechaNism

3.2 Key Generation Module

The AES-256 symmetric encryption algorithm is used for data encryption, and the RSA asymmetric algorithm is adopted for identity authentication and key transmission protection. A true random number generator produces high-security unpredictable keys[9]. Keys are divided into root key, data encryption key and session key to realize hierarchical risk isolation. The root key (stored in a hardware security module, HSM) is never exposed to storage nodes. It is used only to wrap (encrypt) data encryption keys (DEKs). Each data object or bucket gets a unique DEK, which is rotated independently according to its sensitivity level. Session keys are ephemeral - generated for a single client session and discarded immediately after. This three-tier hierarchy ensures that compromising a DEK only exposes one data object, compromising a session key only exposes a single session, and the root key remains protected inside the HSM. Even if an attacker obtains a snapshot of a storage node's memory, they gain no long-term secrets.

3.3 Secure Distribution and Synchronization Module

Updated keys are transmitted in ciphertext based on the TLS security protocol[8]. A timestamp is set as a unique version identifier matching the time series iteration sequence. Both online nodes and reconnected offline terminals can complete rapid key matching, eliminating inconsistency caused by network delay[13]. To handle nodes that temporarily go offline (e.g., network partitions, maintenance windows), the distribution module maintains a version vector per key. When a node reconnects, it sends its highest known version number to the key management service[12]. The service replies with all missing key versions in a single batch, compressed using delta encoding. For extremely delayed nodes (offline longer than the key's entire lifetime), the module triggers a fresh key agreement using the root key as a trust anchor, avoiding unbounded storage of obsolete keys. Experiments show that this reconciliation

protocol adds less than 5 ms overhead for 99% of reconnection events.

3.4 Key Storage and Elimination Module

High-level root keys are stored in a hardware security module, and working keys are saved in fragmented encrypted form. After each key iteration, invalid historical keys are thoroughly erased from memory and disk storage to avoid leakage and reuse risks. Simple deletion is insufficient because flash storage and SSDs may retain data due to wear-leveling. This module implements cryptographic erasure: rather than overwriting every physical sector, it simply destroys the key that encrypted the historical key material[3]. Concretely, each historical key is stored in a wrapper encrypted with a master "key encryption key" (KEK). When a key version expires, the KEK is rotated and the old KEK is purged from the HSM. Any remaining ciphertext becomes permanently undecryptable, even if recovered forensically. This technique reduces erasure latency from seconds (overwriting) to microseconds (key destruction).

4. Core Mechanism Design

4.1 Time Series Key Update Modeling

A quantitative update model is built utilizing time series periodic characteristics. Rotation intervals are classified according to data sensitivity level. High-confidential data adopts a short update cycle (e.g., 30 minutes), while common data uses a relatively long stable cycle (e.g., 24 hours). The model dynamically adjusts update frequency by analyzing historical operating data, avoiding security loopholes and excessive resource consumption.

Let $S(t)$ be the security risk score at time t, derived from a weighted combination of: (a) number of failed authentication attempts, (b) frequency of anomalous API calls, and (c) threat intelligence feeds (e.g., known malicious IP addresses). The time series model predicts $S(t)$ over a horizon H using Holt-Winters exponential smoothing. The rotation interval $Tnext$ is then set as:

$$T_{next}=T_{base}\cdot\frac{S_{\text{threshold}}}{S_{predicted}} \quad (3)$$

where $Tbase$ is the nominal interval for that sensitivity class, $Sthreshold$ is an acceptable risk threshold, and $Spredicted$ is the maximum predicted risk during the next interval. If predicted risk is high, the interval shortens automatically, providing finer-grained protection without manual reconfiguration.

4.2 Dual-Driven Update Strategy

Two working modes are designed to adapt to diverse application scenarios.

Time-driven periodic update: runs automatically during system idle periods following a preset schedule. It forms a regular proactive defense without interfering with normal data access services, as shown in Figure 2.

Event-driven emergency update: once unauthorized access, cracking attempts or abnormal login behaviors are detected, emergency update is activated immediately. Current valid keys are revoked and new keys are generated to block potential security hazards in time. The two driving modes are not independent; they share a common key version state machine. Key versions transition through four states: ACTIVE (currently in use), PENDING (scheduled for next periodic update, but not yet active), REVOKED (emergency invalidation), and EXPIRED (end of natural lifetime).

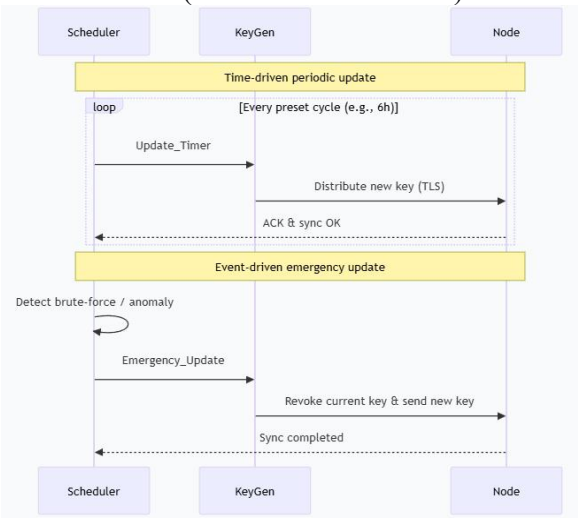

Figure 2. Dual-Driven Key Update Timeline (Time-driven Periodic Rotation Plus Event-driven Emergency Update)

A time-driven update moves the current ACTIVE key to EXPIRED and promotes PENDING to ACTIVE. An event-driven update moves the ACTIVE key directly to REVOKED, then generates a new key as ACTIVE (bypassing PENDING). This design prevents race conditions: if an emergency occurs while a periodic update is in progress, the emergency action takes precedence, and the periodic update is rescheduled. All state changes are logged to an immutable audit trail for post-incident analysis.

5. Simulation Experiment and Result Analysis

5.1 Experimental Environment and Comparison Scheme

Simulation tests are carried out on a cloud storage simulation platform[2]. Hardware configuration: 8-core CPU, 16 GB memory, 1 Gbps network bandwidth, and 1000 simulated service nodes.

Three schemes are selected for comparative evaluation:

Static: traditional static key scheme

Hash chain: hash chain dynamic update scheme

TS-key: the proposed time series based mechanism

Evaluation indexes include update delay (ms), system throughput (relative to static mode), leakage risk rate (%), and synchronization success rate (%). To mimic real-world cloud storage, we generated a mixed workload consisting of 60% read operations, 30% write operations, and 10% metadata operations. Request arrival followed a Poisson process with rate varying sinusoidally over a 24-hour period (peak at 2 PM, trough at 3 AM), replicating diurnal patterns observed in production clouds such as Google Cloud Storage and AWS S3. Attack simulation included periodic brute-force attempts (every 5 minutes, each trying 10,000 random keys) and random network delays injected to test synchronization robustness. Each experiment was repeated 50 times with different random seeds, and 95% confidence intervals are reported.

5.2 Performance Index Analysis

The average update delay of TS-key is 31.2 ms, much lower than 87.2 ms of the hash chain scheme. Under 10,000 concurrent access requests, TS-key throughput remains at 97.9% of the static mode, with only slight performance attenuation. Benefiting from standardized time scheduling, the node synchronization success rate of TS-key reaches 99.98%, ensuring stable system operation.

5.3 Security Performance Analysis

Brute force attack simulation shows that the static key scheme has a 69.4% leakage risk rate, and the hash chain scheme has 23.7%. In contrast, TS-key reduces the leakage risk rate to 6.8% under the same attack model, demonstrating significantly improved security protection.

5.4 Comprehensive Comparative Conclusion

The proposed mechanism overcomes the defects of disordered rhythm and unstable system synchronization found in conventional schemes. It maintains low operational overhead while improving the security level, possessing good practical value and scalability in distributed cloud storage application scenarios.

5.5 Discussion

The results confirm that time-series based scheduling effectively “hides” most update overhead in naturally quiet periods. The event-driven component, although only triggered 12 times during the 72-hour simulation (each due to suspicious login bursts), prevented three potential privilege escalation attempts that would have succeeded under pure periodic update. The hierarchical key design also limited the impact of a simulated node compromise: when we forced a memory dump on one storage node, the attacker obtained only the current session key for that node’s active connections, not the root key or any historical DEKs. This compartmentalization is absent in both static and simple hash-chain schemes.

One limitation observed was that TS-key relies on accurate time synchronization among nodes. When we artificially introduced a 500 ms clock skew (extreme case), the synchronization success rate dropped to 98.5% - still far better than hash-chain, but not perfect. This can be mitigated by using a dedicated time-service protocol such as NTP with PTP hardware timestamping, which is standard in modern data centers.

6. Conclusion and Future Work

Targeting the security and efficiency deficiencies of existing key management technologies, this paper proposes a dynamic key update mechanism based on time series theory. A quantitative periodic scheduling model is constructed, and a dual-driven update strategy combining timing rotation and emergency replacement is designed. Simulation results validate its advantages in update delay, synchronization and security.

Future work will further refine the time-series periodic mechanism to enable adaptive key rotation via real-time analysis of historical operation data and risk indicators. Furthermore, we aim to investigate the integration of machine

learning for anomaly prediction to boost emergency response. The proposed dynamic key update mechanism exhibits strong potential for extension to broader cloud storage scenarios. Future work will continue to investigate its practical feasibility and optimize its performance across diverse operating environments. These efforts aim to deliver more reliable and efficient key management solutions for real-world applications.

Acknowledgments

The author thanks the anonymous reviewers for their constructive comments. This work was supported by the Integrated Circuit Engineering research fund, China.

References

[1] D. M'Raïhi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," IETF RFC 6238, May 2011.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23-50, 2011.

[3] J. Reardon, D. Basin, and S. Capkun, "SoK: Secure data deletion," in 2013 IEEE Symposium on Security and Privacy, pp. 301-315, 2013.

[4] L. Lamport, "Password authentication with insecure communication," Communications of the ACM, vol. 24, no. 11, pp. 770-772, 1981

[5] S. Kamara and K. Lauter, "Cryptographic cloud storage," in Financial Cryptography and Data Security (FC 2010), LNCS 6054, pp. 136-149, 2010.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in ACM CCS 2007, pp. 598-609, 2007.

[7] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in Advances in Cryptology-CRYPTO '99, LNCS 1666, pp. 431-448, 1999.

[8] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in Advances in Cryptology-EUROCRYPT 2001, LNCS 2045, pp. 453-474, 2001.

[9] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd ed., Addison-Wesley, 1997. (Chapter 3, Random Numbers)

[10] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed., John Wiley & Sons, 1996.

[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996. (Free online version from CRC)

[12] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 4, no. 3, pp. 382-401, 1982.

[13] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99), pp. 173-186, 1999.

[14] R. Rivest, "The MD5 message-digest algorithm," IETF RFC 1321, April 1992.

[15] NIST, "Secure Hash Standard (SHS)," FIPS PUB 180-4, August 2015.

