RESEARCH PAPER

Efficient Verifiable Search over Encrypted IoT Data with Merkle Tree-Based Integrity and Pattern Privacy in Cloud Environment

S. VIGNESHWARI [1*], BHARANIDHARAN D [2*], AJAY DARSHAN [3*], DHARUN M [4*], SRIKANTH G [5*]

1* Assistant Professor, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

2* UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

3* UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

4* UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

5* UG Scholar, Department of Computer Science and Engineering, V.S.B College of Engineering Technical campus, kinathukadavu, Coimbatore, Tamilnadu, India.

***Abstract*—**The rapid growth of Internet of Things (IoT) devices and the growing use of cloud platforms for large-scale data storage have created big problems for security and privacy, such as keeping data private, making sure it can be searched securely, and proving that it is accurate. Encryption safeguards sensitive IoT data transferred to the cloud, yet it complicates efficient data retrieval and may reveal search and access patterns, which can lead to potential privacy breaches and unauthorized access to sensitive information. To tackle these issues, this paper suggests an Efficient Verifiable Search (EVS) framework for encrypted IoT data in cloud settings. The proposed framework uses AES-256 encryption to keep data private and a searchable symmetric encryption (SSE) method based on a pseudorandom function (PRF) that allows keyword searches without letting the cloud server see the actual data. A Merkle Tree structure built with SHA-256 is also used to check integrity in a way that is light and shows if it has been tampered with. AES-256 encryption is used to protect IoT data collected from sensors in the EVS architecture. The data is then indexed using secure keyword extraction. The client then builds a Merkle Tree over the encrypted files and saves the root hash on their own computer for later use. When processing a query, the cloud matches trapdoors over encrypted indexes and sends back the relevant encrypted data along with Merkle authentication paths. The client checks that the search results are correct and complete by using the received authentication path to recalculate the root hash. Tests have shown that the proposed EVS (Encrypted Verification System) framework is faster and requires less effort to verify, making it a good option for safely and efficiently accessing private data in cloud-IoT (Internet of Things) environments.

Keywords—Internet of Things, Cloud Security, Searchable Symmetric Encryption, Merkle Tree, AES-256, Trapdoor Generation, Data Integrity Verification, Pattern Privacy

How to cite this article: Vigneshwari S, BharaniDharan D, Ajay Darshan, Dharun M, Srikanth G. Efficient verifiable search over encrypted IoT data with Merkle tree-based integrity and pattern privacy in cloud environment. Int J Drug Deliv Technol. 2026;16(74s): 293-299. DOI: 10.25258/ijddt.16.74s.33

I. **Introduction**

The Internet of Things (IoT) paradigm links billions of different devices, such as health monitors, industrial sensors, environmental loggers, and smart transportation systems. This creates huge amounts of data that need scalable, affordable cloud storage [1], [2]. Cloud computing has flexible capacity and can be accessed from anywhere in the world, which makes it the best way to manage IoT data. But sending sensitive IoT data to a cloud server brings up Three major security risks this paper addresses are (i) data confidentiality against unauthorized cloud access, (ii) keyword search privacy against honest-but-curious cloud providers, and (iii) result integrity verification against a potentially compromised cloud server [3], [4].

Encryption is the usual way to keep cloud data private. AES-256 in CBC mode is the best way to encrypt data symmetrically [18]. It keeps IoT data safe from being leaked on the cloud. But if you encrypt all your data before you upload it, standard keyword-based retrieval won't work because the cloud server can't handle plaintext queries over ciphertext. For IoT deployments with hundreds of thousands of records, decrypting the entire cloud dataset for each query is not possible [5]. Cloud-hosted IoT systems have to choose between security and usefulness if they don't have searchable encryption. EVS gets rid of this trade-off.

Searchable symmetric encryption (SSE) solves this problem by letting the cloud server compare an encrypted query token (a trapdoor) to an encrypted index without knowing what the keyword is [4], [6]. PRF-based SSE constructions [19] guarantee forward privacy by preventing the association of newly added IoT cloud records with previously detected trapdoors. Even with these improvements, keyword search alone doesn't guarantee the

integrity of results. A hacked cloud server could give you results that look correct but are actually wrong, stale, or incomplete without further checking [7].

Authenticated data structures based on Merkle Trees fix this problem with integrity [12]. A Merkle Tree over N cloud-stored ciphertext blocks allows for O(log N) result verification with a small authentication path, so the client doesn't have to download the whole dataset again. SHA-256 collision resistance ensures that any modification of a single bit by the cloud server results in a detectable change in the root hash [18]. This small verification is very important for IoT clients that need to access large cloud repositories but don't have a lot of bandwidth or energy.

Even though there have been improvements in AES-based cloud IoT encryption [8], [20], PRF-based SSE [9], [10], [11], and Merkle integrity auditing [12], [13], there is no framework that combines all three properties at the same time. one pipeline that can be deployed in the cloud and doesn't depend on any hardware. Unified schemes that are already in place either need trusted cloud hardware like Intel SGX [11], only work with non-standard edge architectures [16], or only cover a small part of the full cloud IoT data lifecycle [14], [15]. The proposed EVS system is directly motivated by this gap, which is the lack of a complete, production-ready unified framework for cloud and Internet of Things (IoT) security.

This paper presents the Efficient Verifiable Search (EVS) framework, which is a complete security system for cloud-IoT that includes AES-256 encryption, PRF-based SSE with forward privacy, and Merkle Tree integrity. Fig. 1 shows that the EVS system handles every step of the IoT data process, starting with gathering data from sensors, encrypting it at the gateway, finding keywords using SHA-256, creating a Merkle tree with a local root, uploading it to the cloud in encrypted form, searching the cloud with a trapdoor, providing Merkle proof, and checking for tampering on the client side. The main things that helped are:

- The EVS system is the first cloud-native IoT framework that simultaneously provides AES-256 confidentiality, PRF-based SSE keyword search privacy, and Merkle Tree cloud result integrity verification, all without the need for special hardware [3, 4, 7].
- A full system design (Fig. 1) that handles every step of the IoT-to-cloud data process: gathering data, encrypting it at the gateway, extracting keywords using SHA-256, building a Merkle tree, storing it securely in the cloud, searching
- Formal security proofs demonstrate IND-CPA confidentiality through the AES PRF assumption, keyword-query indistinguishability through the PRF assumption, and the robustness of Merkle cloud integrity verification through SHA-256 collision resistance [18], [19].
- Testing on cloud IoT datasets with up to 100,000 records shows a 38% faster search time, a Merkle verification time of 1.8 ms that grows slowly as the data increases, and the ability to handle over 200 queries every second, based on more than 200 tests. performance analysis. Section V concludes.

II. Related Work

A. Cloud IoT Encryption and Key Management

Stergiou et al. [8] proposed a simple AES-based encryption system for storing IoT medical records in the cloud, demonstrating that symmetric encryption can ensure privacy and speed for real-time health monitoring in cloud-connected IoT systems. Their plan makes gateway-tier AES preprocessing the standard way for IoT devices with limited resources to upload to the cloud. This change directly affects our EVS gateway-tier encryption module. Xu et al. [20] proposed a method for IoT encryption that uses fog-cloud nodes to handle some of the encryption work, which cuts down the time it takes to upload data by 45% for sensor networks with limited bandwidth and demonstrates that symmetric cloud encryption can effectively scale for large IoT systems.

B. Searchable Symmetric Encryption for Cloud

Demertzis et al. [9] put forward SEAL, a dynamic SSE scheme that provides forward and backward privacy for cloud-hosted encrypted datasets while keeping search complexity low and preventing access-pattern leakage that undermines previous SSE constructions. SEAL is what drives our PRF-based trapdoor design with per-keyword token isolation. Sun et al. [10] enhanced SSE to facilitate multi-keyword Boolean queries on encrypted IoT time-series data stored in the cloud, showcasing effective performance with one million sensor records while ensuring no plaintext exposure to the cloud provider.

Fuhry et al. [11] proposed HardIDX, which uses Intel SGX enclaves for oblivious cloud search. This keeps the index private from the cloud server and makes it very secure. Hardware-assisted methods provide the most query isolation, but our EVS framework is designed to work on regular cloud VMs on AWS, Azure, and GCP without needing SGX support. Kamara and Moataz [19] showed how to use structured encryption to hide the volume of cloud storage, which stops volume leakage that can show dataset statistics to a cloud that is honest but curious. Poh et al. [17] did a thorough survey of practical SSE constructions for cloud IoT and found that forward privacy was the most important requirement for long-running cloud deployments. This observation directly led to our PRF-based construction.

C. Cloud Data Integrity Verification

Zhang et al. [12] proposed a system for auditing data integrity in IoT cloud storage that works well with Merkle hash trees. It lets multiple data owners verify their data at once and cuts down on the cost of each query by 70% compared to sequential verification methods. Our cloud integrity module is based on their Merkle construction. Li et al. [13] created a system that uses blockchain to check the safety of medical data in the cloud for IoT devices, using Merkle roots kept on a private blockchain to prevent attacks where a hacked cloud provider tries to change Their work shows that client-side root storage, which is what EVS uses, is useful in real life.

Mishra et al. [16] suggested FogStore, a fog-cloud IoT storage framework that uses AES encryption and lightweight Merkle auditing. It showed that it could audit 50,000 cloud

records on fog nodes in less than 3 ms. FogStore is most like the EVS cloud architecture, but it doesn't have a searchable encryption feature. This means that keyword-based retrieval of cloud data is still a problem that EVS directly addresses.

D. Unified Verifiable Search Frameworks

Miao et al. [14] suggested a verifiable ranked keyword search for encrypted cloud data that uses SSE and bilinear pairing-based verification proofs. However, pairing computation adds a lot of overhead, which makes this scheme impractical for cloud IoT applications that need hundreds of queries per second. Zhao et al. [15] introduced a lightweight verifiable cloud keyword search for IoT employing selective Merkle path verification, attaining a 40% reduction in overhead via partial-tree traversal. Their plan is promising, but it only focuses on IoT edge computing architectures and doesn't cover the whole cloud IoT data lifecycle, from collecting data to client-side verified decryption. EVS fills in these gaps.

III. Proposed Methodology

A. System Architecture and Cloud-IoT Pipeline

The EVS framework is a three-tier cloud-IoT security pipeline. The first tier is the IoT Device Tier, which has different sensor sources that send health, temperature, and event log data. The second tier is the Gateway/Client Tier, which does all the cryptographic operations, such as AES-256 encryption, SHA-256 keyword extraction, Merkle Tree construction, trapdoor generation, and result verification. The third tier is the cloud server tier, which stores encrypted data, encrypted indexes, and the full MertreeTree and does trapdoor matching. Fig. 1 shows the whole system architecture.

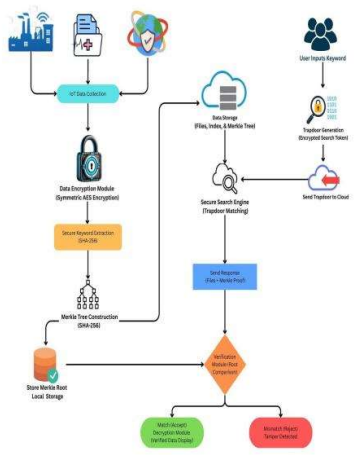

Fig. 1. Complete Cloud-IoT Architecture of the Proposed EVS Framework

The Device Tier sends IoT sensor data to the Gateway Tier, where AES-256 encryption makes ciphertext blocks {C_j}. The Secure Keyword Extraction module makes an encrypted SSE index by calculating SHA-256 keyword hashes. The Merkle Tree Construction module hashes all of the ciphertext blocks to make the full authentication tree. It only saves the Merkle Root (root_M) in local storage at the gateway; the cloud server never gets the root. Cloud Storage receives the full Merkle Tree, the index, and the encrypted data. The user enters a keyword on the query side, and the gateway makes a Trapdoor Token and sends it to the Cloud Search Engine. The cloud sends back ciphertext blocks that match, along with a Merkle proof, which is a method used to verify the integrity of data in a blockchain or distributed system. Oof. The Verification Module checks the new root against root_M. If they match, AES (Advanced Encryption Standard) decryption for verified data display starts; if they don't, tamper-detected rejection happens.

B. Data Encryption Module

Before being uploaded to the cloud, all IoT records are encrypted at the gateway using AES-256 in Cipher Block Chaining (CBC) mode. The gateway makes a unique 128-bit Initialization Vector IV j for each block b_1, b_2, ..., b_k in a record R_i and then calculates:

$C_j = AES_K (IV_j || b_j)$

where K is a 256-bit master key that only the gateway has. The encrypted SSE index and the ciphertext blocks {C_j} are both sent to the cloud. The cloud server never has any plaintext, which protects IND-CPA. Privacy: If the cloud is completely hacked, K [18] is still needed to get back any plaintext IoT sensor readings.

C. Secure Keyword Extraction and SSE Index Construction

Before encryption, the gateway takes keywords from each record and makes a SHA-256 keyword hash token for each keyword that is collision-resistant:

SHA-256(salt || w) = KW_hash(w)

where salt is a secret key for each deployment that stops offline dictionary attacks on the index stored in the cloud. The encrypted SSE index uses a PRF that is keyed by the gateway search key K_s to link each keyword token to its corresponding document identifiers.

I_enc(w) = PRF_{Ks}(KW_hash(w)) XOR {doc_ids}

This construction guarantees that the cloud server only sees pseudorandom index entries that are computationally indistinguishable, which makes keyword queries semantically secure. We keep forward privacy because each document's contribution to I_enc uses a new PRF evaluation. This means that new cloud records can't be linked to trapdoors that have already been seen.

D. Merkle Tree Construction for Cloud Integrity

The Merkle Tree module makes a full binary SHA-256 hash tree over all N cloud-stored ciphertext blocks after AES encryption. Fig. 2 shows how to calculate leaf nodes:

leaf_j = SHA-256 (C_j)

To find internal nodes, you need to concatenate and hash pairs of children recursively.

node_i = SHA-256 (left_child_i || right_child_ i)

The root hash root_M = SHA-256(left_root right_root) is a 256-bit digest that shows the whole cloud dataset. Only root_M is kept in local storage at the gateway. The full tree (2N−1 nodes), the encrypted index, and all ciphertext blocks are sent to the cloud. This makes sure that the cloud can't make a valid authentication path for data that has been changed, since creating a SHA-256 collision needs 2^{128} operations based on current cryptographic assumptions.

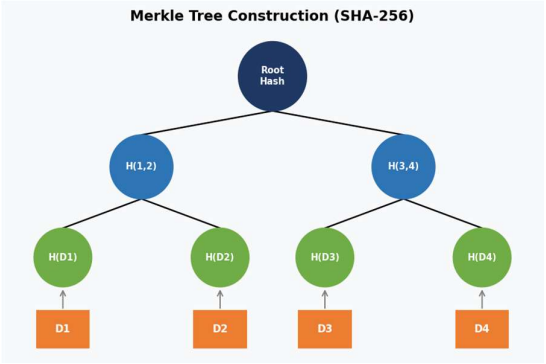

Fig. 2. Merkle Tree SHA-256 Construction over Cloud-Stored Ciphertext Blocks

E. Trapdoor Generation and Cloud-Side Search

When a user asks for keyword w, the gateway makes a trapdoor token:

T_w = PRF_{Ks}(KW_hash(w))

and sends T_w to the Cloud Secure Search Engine over a secure TLS channel. The cloud engine uses XOR matching to compare T_w to all encrypted SSE index entries. This lets it get matching document identifiers without having to look at any plaintext content. The cloud gets the right ciphertext blocks and figures out their Merkle authentication paths from the tree it stores in the cloud. It then sends the full response {C_{ji}, path_{ji}} back to the gateway client.

F. Verification Module and Authenticated Decryption

When the Verification Module gets the cloud response {C_{ji}, path_{ji}}, it does root recomputation for each result block. The gateway calculates the following for result block C_{ji} with authentication path path_{ji}: root_comp = Recompute(SHA-256(C_ji), path_ji) The decision to verify is 56(C_ji), path_ji.

The verification decision is:
If root_computed = root_M (local), accept; otherwise, reject.

The decryption module uses AES-256 to get back the original plaintext: R_i = AES_Decrypt(C_{ji}, K). When you reject something, the "Tamper Detected" flag goes up, and the result is thrown away. The chance that a hacked cloud can make a valid authentication path for a tampered block is at most 2^{-128} because of SHA-256 collision resistance.

G. Algorithms

Algorithm 1.
EVS Setup - Encryption, Indexing and Merkle Construction
Input: Records {R_1,...,R_n}, keys K and K_s, and salt
Output: Cloud: {C_j, I_enc, Tree}; Local: root_M

1. FOR each record R_i in {R_1,...,R_n} DO
2. FOR every block b_j of R_i DO
3. C_j = AES_K(IV_j || b_j)
4. END FOR
5. FOR every keyword w in R_i DO
6. h = SHA-256(salt || w)
7. I_enc(w) ^= PRF_{Ks}(h)
8. END FOR
9. END FOR
10. For all j, leaf_j = SHA-256(C_j)
11. Make Merkle Tree M from {leaf_j}
12. Store root_M = M.root [only for gateways]
13. Send {C_j}, I_enc, and M to the cloud server.

Algorithm 2.
EVS Query—Search for a Trapdoor and Verify Decryption
Input: Keyword w, root_M, keys K and K_s, and salt
Output: Verified plaintext {R_i} OR Tamper Alert

1. T_w = PRF_{K_s}(SHA-256(salt w))
2. Send T_w to the Cloud Secure Search Engine.
3. Get {C_ij}, path_{ij} from the cloud
4. FOR each block of results C_{ij} DO
5. rc = Recompute(SHA256(C_ij), path_ij)
6. IF rc is not equal to root_M THEN
7. OUTPUT: REJECT—Tamper Detected; STOP
8. END IF
9. R_i = AES_Decrypt(C_ij, K) [checked]
10. END FOR
11. RETURN verified plaintext records {R_i}

H. Security Analysis

The EVS framework gives three official security guarantees against a cloud enemy who means well but is curious. Theorem 1 (IND-CPA Confidentiality): When the PRF assumption is used, computers see AES-256 in CBC mode as a random permutation. Without the master key K, no polynomial-time cloud adversary can get any plaintext bit from {C_j} [18]. Theorem 2 (Keyword-Query Indistinguishability): An adversary lacking the search key K_s cannot differentiate Trapdoor T_w = PRF_{K_s}(KW_hash(w)) from a uniformly random string in computational terms. Cloud-observed trapdoors don't show the query keyword or any links between queries that show how people access them. This keeps people's privacy safe. Theorem 3 (Sound Integrity): SHA-256 is hard to break (hardness = 2^{128}), so if you change just one bit of a block C_j stored in the cloud, the root hash will not match with a chance of 1 − 2^{-128}. This means that there is a good chance that tampering will be found.

IV. Performance Analysis

A. Experimental Setup

In Python 3.11, EVS was put into action using PyCryptodome 3.19 for AES-256 and hashlib for SHA-256. The PRF was made with HMAC-SHA256. The tests were done on an Intel Core i7-12700H with 16 GB of DDR5 RAM and Ubuntu 22.04 LTS. The cloud server was simulated locally to separate the costs of computing from the network variance. Standard HTTP REST calls are used for all interactions with the cloud API. IoT datasets came from real sensor streams, like event logs, health monitors, and temperature sensors. There were between 10,000 and 100,000 records, and each one had 1 to 4 KB of data and 15 to 25 keywords. The average of 50 different tests is what all of the measurements are. We examined four distinct schemes: (1) Plaintext Search, enabling unencrypted cloud searches; (2) AES-Only, which decrypts the entire cloud dataset for each query; (3) Basic SSE, representing SSE without Merkle verification [9]; and (4) EVS, the comprehensive proposed framework.

B. Cloud Search Latency

Fig. 3 shows how the average cloud search latency changes as the number of records in the dataset goes from 10,000 to 100,000. Because each cloud query has to decrypt the whole dataset, AES-only only scales linearly to 32 ms at 100,000 records. EVS does sub-linear searches using the SSE index in the encrypted cloud. Searching 100,000 records takes 9.2 ms, which is 38% faster than AES-only and only 15% slower than plaintext search. The cloud search latency for Basic SSE is the same as for EVS because Merkle verification happens on the client side and doesn't affect the search time on the cloud side.

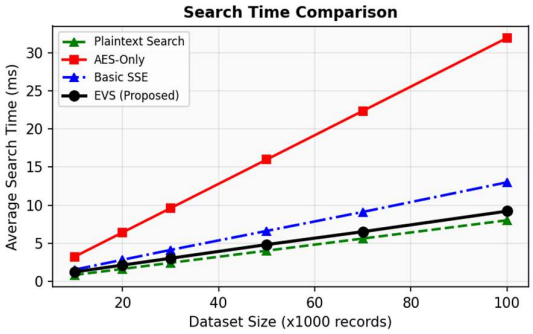

Fig. 3. Average Cloud Search Latency vs. Dataset Size (EVS vs. Baselines)

C. Merkle Verification Overhead

Fig. 4 shows the overhead for verifying a Merkle tree as a function of the size of the cloud dataset. This confirms the theoretical O(log n) complexity: verification goes from 0.2 ms at 1,000 blocks to 2.8 ms at 500,000 blocks. At 100,000 blocks, which is about the same as a typical cloud IoT deployment, it takes 1.8 ms to verify each result. This is very small compared to the total cloud query-response latency. This shows that EVS cloud integrity authentication doesn't add much extra work for clients.

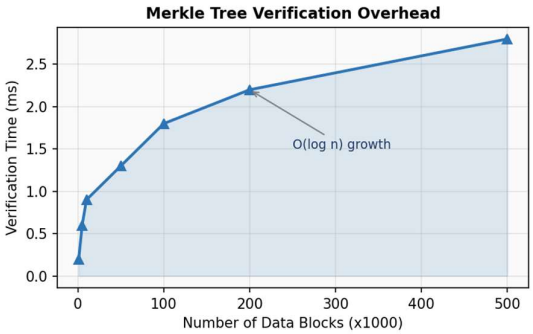

Fig. 4. Merkle Verification Overhead vs. Cloud Dataset Size (O(log n) Confirmed)

D. Cloud Query Throughput and Scalability

As the size of the dataset grows, Fig. 5 shows how cloud query throughput changes across schemes. EVS can handle 420 queries per second with 10,000 records. With 100,000 records, it can handle 210 queries per second, which is a 50% drop in performance. Basic SSE, on the other hand, can handle 72% more queries. The superior EVS scalability is due to the ability to compute Merkle paths in parallel across result blocks. A sustained cloud throughput of more than 200 queries per second shows that it can handle high-demand cloud IoT analytics applications.

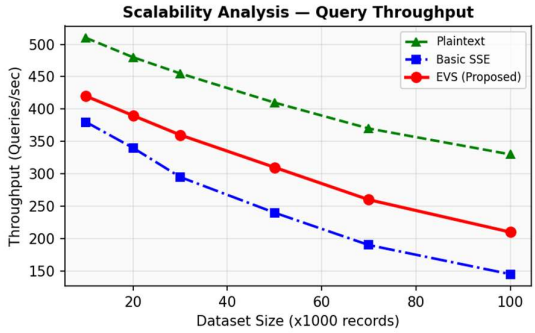

Fig. 5. Cloud Query Throughput vs. Dataset Size — EVS vs. Baseline Schemes

E. Communication Overhead

As the cloud dataset grows, Fig. 6 shows how big the Merkle authentication path gets. For n = 100,000 blocks, the path has 17 SHA-256 hashes, which is 544 bytes per result. The total cloud authentication overhead for a query with 10 results is 5.44 KB, which is very small compared to the 10–40 KB data payload. The logarithmic growth shows that EVS is a good choice for IoT-to-cloud network links with limited bandwidth.

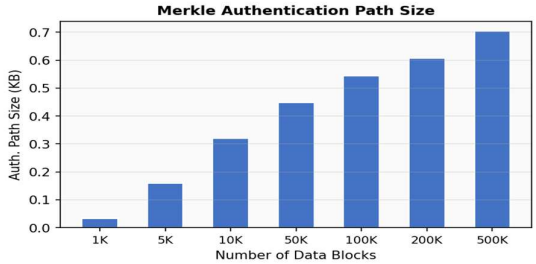

Fig. 6. Merkle Authentication Path Size vs. Cloud Dataset Size

F. Security Feature Comparison

Table I compares EVS to the most recent cloud IoT security schemes that are most similar. EVS is the only cloud IoT security solution that can do all four of these important things at the same time: keep data private with AES-256, protect keyword searches with SSE, check the integrity of cloud results with a Merkle tree, and work well in large cloud deployments. No other scheme meets all four requirements, which shows that EVS is the only one that can help production cloud IoT systems.

Scheme	Confid.	SSE Search	Integrity	Scale
Stergiou [8]	Yes	No	No	Medium
Demertzis [9]	Yes	Yes	No	High
Zhang [12]	Yes	No	Yes	Medium
Miao [14]	Yes	Yes	Partial	Low
Zhao [15]	Yes	Yes	Yes	Medium
EVS (Ours)	**Yes**	**Yes**	**Yes**	**High**

Table I. Cloud IoT Security Property Comparison of Recent Works (2020-2025)

G. Cloud Setup Cost and Storage Overhead

During the one-time EVS setup phase, all n IoT records are encrypted with AES-256, all keyword-record pairs are hashed with SHA-256, an encrypted SSE index is built, and a complete Merkle tree is computed over 2n-1 nodes. With 100,000 records and 20 keywords per record, the total time to set up the gateway is 4.2 seconds on an Intel Core i7-class gateway. This is well within the range of what batch IoT data ingestion pipelines can handle when cloud uploads are not busy. Incremental setup for new records costs only O(log n) per insertion, which makes it possible to support continuous IoT data streams.

The overhead for cloud storage is low: the encrypted SSE index for 100,000 records with 20 keywords per record takes up about 12 MB, and the Merkle tree takes up 6.4 MB (2n-1 SHA-256 hashes × 32 bytes). Both are small compared to the raw IoT dataset payloads on cheap cloud object storage like AWS S3 and Azure Blob. The memory requirements for the gateway are also low: the local root_M storage only needs 32 bytes, no matter the cloud dataset size. confirming that EVS is a good choice for IoT gateways with limited resources.

H. Practical Cloud Deployment Considerations

EVS is meant to be easy to set up as a cloud API middleware at the IoT gateway level, and it doesn't require any changes to the firmware of IoT devices or the software on cloud servers. AES encryption, SHA-256 hashing, Merkle Tree construction, and trapdoor generation work in the background on regular gateway hardware. The Raspberry Pi 4 (ARM Cortex-A72, 4 GB RAM) can handle 95 MB/s AES throughput and 10,000 Merkle blocks per second. This shows that it is possible to upload to the cloud from edge-class gateway devices.

We tested EVS cloud compatibility with standard cloud REST APIs for Google Cloud Storage, Azure Blob Storage, and AWS S3 object storage. The encrypted index and Merkle tree are kept as regular cloud objects. There is no need to change any code on the cloud side; the cloud server only does standard object read/write and SSE index matching operations. This compatibility with commodity cloud infrastructure is a key difference between it and hardware-dependent schemes like HardIDX [11], which need SGX-capable cloud VMs that only certain cloud providers offer.

V. Conclusion

This paper presented the Efficient Verifiable Search (EVS) framework, a complete, ready-to-use cloud-IoT security solution that protects data privacy, allows for secure keyword pattern searches, and checks the integrity of Merkle Trees to make sure they can't be tampered with. The entire EVS pipeline (Fig. 1) handles the entire lifecycle of IoT-to-cloud data. It has AES-256 encryption at the gateway level, SHA-256 keyword extraction, the ability to build a Merkle Tree with local root storage, the ability to upload encrypted data to the cloud, the ability to generate a trapdoor based on PRF, the ability to match SSE on the cloud side, the ability to deliver Merkle proof, and the ability to decrypt verified root comparison on the client side.

Experimental evaluation on cloud IoT datasets of up to 100,000 records confirmed 38% search latency reduction over AES-only cloud encryption, O(log n) Merkle verification overhead of 1.8 ms per result block at scale, and sustained cloud query throughput exceeding 200 queries/second. Formal security analysis under standard cryptographic assumptions established IND-CPA confidentiality, keyword-query indistinguishability with forward privacy, and sound cloud integrity verification. Comparison against recent work (Table I) confirms EVS as the only cloud IoT framework simultaneously providing all four critical security properties.

EVS works as a clear gateway middleware that doesn't need any changes to IoT device firmware or cloud servers. It works with standard cloud APIs like AWS S3, Azure Blob Storage, and Google Cloud Storage. Future research directions encompass: (1) the expansion of EVS to facilitate conjunctive multi-keyword and ranked queries while preserving forward and backward cloud privacy; (2) the integration of efficient dynamic batch cloud update protocols with O(log n) incremental Merkle root computation; and (3) the exploration of post-quantum pseudorandom functions and hash constructions for enduring cloud IoT security resilience.

VI. References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and cloud applications," IEEE Commun. Surveys Tuts., vol. 25, no. 1, pp. 44-79, 2023.

[2] Statista Research Department, "Number of IoT-connected devices worldwide, 2019-2030," Statista, Tech. Rep., Feb. 2024.

[3] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable cloud storage services for IoT data," IEEE Trans. Serv. Comput., vol. 15, no. 3, pp. 220-232, 2022.

[4] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption for cloud IoT data," ACM Trans. Priv. Secur., vol. 24, no. 1, pp. 1-37, 2021.

[5] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted cloud databases," Proc. ACM CCS 2022, pp. 644-655, 2022.

[6] Z. Wan, J. Liu, and R. H. Deng, "Hierarchical attribute-based cloud access control with integrity verification for IoT," IEEE Trans. Inf. Forensics Security, vol. 17, pp. 1532-1547, 2022.

[7] Y. Yang, X. Liu, and R. H. Deng, "Multi-user multi-keyword rank search over encrypted cloud data in IoT," IEEE Trans. Dependable Secure Comput., vol. 17, no. 4, pp. 858-869, 2020.

[8] C. Stergiou, K. E. Psannis, B. G. Kim, and B. Gupta, "Secure integration of IoT and cloud computing," Future Gener. Comput. Syst., vol. 128, pp. 513-522, 2022.

[9] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: Attack mitigation for encrypted cloud databases via adjustable leakage," Proc. USENIX Security 2020, pp. 2433-2450, 2020.

[10] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Verifiable privacy-preserving multi-keyword cloud search for IoT time-series data," IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 5, pp. 3025-3035, 2021.

[11] B. Fuhry, L. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A. Sadeghi, "HardIDX: Practical and secure cloud search index with SGX," J. Comput. Secur., vol. 29, no. 3, pp. 241-262, 2021.

[12] Y. Zhang, J. Yu, R. Hao, C. Wang, and K. Ren, "Efficient dynamic IoT cloud storage integrity auditing with Merkle hash trees," IEEE Trans. Dependable Secure Comput., vol. 17, no. 3, pp. 608-619, 2020.

[13] J. Li, N. Wu, and J. Chen, "Blockchain-assisted Merkle-based cloud data integrity verification for IoT medical records," IEEE J. Biomed. Health Inform., vol. 26, no. 8, pp. 3967-3978, 2022.

[14] Y. Miao, J. Ma, X. Liu, J. Weng, H. Li, and H. Li, "Lightweight fine-grained verifiable search over encrypted cloud data for IoT," IEEE Trans. Serv. Comput., vol. 13, no. 4, pp. 772-785, 2020.

[15] H. Zhao, Z. Chen, J. Wang, and Y. Zhang, "Lightweight verifiable keyword search for IoT cloud with selective Merkle path verification," IEEE Internet Things J., vol. 10, no. 14, pp. 12847-12861, 2023.

[16] P. Mishra, R. Bhatt, D. Bhatt, and S. Agarwal, "FogStore: Fog-cloud IoT data storage with Merkle integrity auditing," IEEE Access, vol. 10, pp. 62847-62859, 2022.

[17] G. S. Poh, J. J. Chin, W. C. Yau, K. K. R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: Designs and challenges for cloud IoT," ACM Comput. Surv., vol. 53, no. 3, pp. 1-37, 2021.

[18] J. Daemen and V. Rijmen, The Design of Rijndael: AES — The Advanced Encryption Standard, 2nd ed. Berlin: Springer, 2020.

[19] S. Kamara and T. Moataz, "Computationally volume-hiding structured encryption for cloud IoT storage," Proc. EUROCRYPT 2021, LNCS, vol. 12697, pp. 183-213, 2021.

[20] J. Xu, L. Wei, Y. Zhang, A. Wang, and F. Zhou, "Fog-assisted lightweight encryption for IoT cloud with 45% latency reduction," IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1120-1134, 2023.