Impact of Post-Quantum Signatures on InnoDB B+-Trees and Efficient Batch Signing

Seung-Won Lee[1 [0009−0009−9583−1387]], Min-Seo Kim[1 [0009−0008−1979−5778]], Ui-Jae Kim[1 [0009−0004−2617−9659]], Hui-Ju Kang[1 [0009−0008−3198−4598]], and Hwa-Jeong Seo[1 [0000−0003−0069−9061]]

Department of Information Computer Engineering,
Hansung University, Seoul, 02876 Korea
dkajdfhd1@gmail.com, msqoo0212@gmail.com, chrisvt424@gmail.com,
neome0919@gmail.com, hwajeong84@gmail.com

Abstract. The transition to post-quantum cryptography (PQC) digital signatures poses an unexpected threat to the storage structure of relational databases. At the same security level, the AIMer-192f signature reaches 13,056 B, which is more than 13 times that of RSA-7680 (960 B). Storing it inline in MySQL InnoDB causes the B^+-Tree fan-out to collapse from the theoretically predicted value of 167 to a measured value of 1. This result experimentally reveals that the off-page storage model in the MySQL official manual has a factor of 167 error in this case. To address this problem, we propose an architecture that combines a split-table schema with a Merkle Tree-based batch signing approach. The proposed architecture ($B = 512$) restores the collapsed fan-out to 41, reduces the number of leaf pages by 97%, and improves insertion throughput by 28.1×. It also reduces the per-document signature storage cost by up to 97.6%. This study quantifies the limitations of the traditional single-table storage approach in a PQC migration environment and presents a practical mitigation architecture.

Keywords: Post-Quantum Cryptography · MySQL InnoDB · Merkle Tree · Dilithium · AIMer

1 Introduction

The emergence of quantum computers and Shor's algorithm has theoretically established that classical public-key cryptosystems such as Rivest-Shamir-Adleman (RSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA) can be broken in polynomial time [16]. In response, the National Institute of Standards and Technology (NIST) finalized the lattice-based digital signature algorithm CRYSTALS-Dilithium as the Module-Lattice-based Digital Signature Algorithm (FIPS 204) and the hash-based signature algorithm SPHINCS+ as the Stateless Hash-based Digital Signature Algorithm (FIPS 205) in 2024 [10]. Korea also selected AIMer as the domestic PQC digital signature standard through the Korea Post-Quantum Cryptography (KpqC) competition [4].

However, practical deployment of PQC digital signatures faces structural challenges. At the same security level, while RSA-7680 produces a 960-byte signature, Dilithium3 produces 3,293 bytes and AIMer-192f reaches 13,056 bytes. Furthermore, once quantum computers become practical, Shor's algorithm could be used to forge RSA and ECDSA signatures, threatening the legal validity of digital documents and the integrity of audit logs. This makes the urgency of transitioning to PQC even greater. Accordingly, the performance overhead of PQC signatures has been actively studied in contexts such as TLS handshakes [14], blockchain [18], and IoT environments [6]. However, existing research has focused on network protocols and distributed systems.

MySQL InnoDB is widely used worldwide as a core data storage infrastructure for services that require integrity guarantees [12]. InnoDB relies on a B^+-Tree clustered index for efficient record retrieval, and its efficiency is determined by the *fan-out*—the number of records a single index page can hold. Since fan-out is inversely proportional to the average row size, storing large PQC signatures inline in the same row as document data causes the row size to grow and the fan-out to drop sharply. In extreme cases, the B^+-Tree can degrade to the level of a linked list with no indexing benefit. Therefore, quantifying the impact of large signatures on the MySQL InnoDB B^+-Tree storage structure during PQC migration and devising countermeasures is an essential research task for a practical PQC transition.

This study is therefore centered on the following key question: "Do large PQC digital signatures cause structural degradation in the MySQL InnoDB B^+-Tree clustered index structure? How severe is this degradation, and how can it be mitigated?" We formulate two specific research questions.

First, how does the signature size of PQC digital signature algorithms affect the fan-out, number of leaf pages and insertion throughput of the InnoDB B^+-Tree? Second, can the combination of a split-table schema and a Merkle Tree-based batch signing mechanism restore the structural efficiency of the B^+-Tree while preserving per-document integrity?

To answer these research questions, this study sets three specific goals that combine theoretical analysis with empirical measurement. First, we measure the fan-out, number of leaf pages, and insertion throughput of RSA-7680, Dilithium3, AIMer-192f, and AIMer-192s under the same conditions. Second, we compare the theoretically predicted B^+-Tree model values with empirical measurements and analyze the causes of any discrepancies. Third, we design a split-table schema that separates signatures from the document table and evaluate its effect on restoring the B^+-Tree fan-out.

To achieve these goals, this study adopts an experimental methodology. We build a controlled benchmark in a MySQL 9.6.0 InnoDB environment with a buffer pool and pages, and insert one million records for each algorithm configuration. B^+-Tree structural metrics are collected through MySQL internal statistics tables, and the relationship between storage cost and insertion performance is analyzed using a batch size of $B = 512$. Detailed experimental settings and implementation tools are described in Section 4.

The main contributions of this study are as follows.

1. We analyze the storage-level impact of large post-quantum cryptographic signatures on MySQL InnoDB clustered indexes, and we experimentally reveal that AIMer signatures can invalidate the expected off-page storage behavior and collapse the B^+-Tree fan-out.
2. We develop an empirical evaluation framework for PQC-signed document storage in InnoDB, measuring how RSA-7680, Dilithium3, AIMer-192f, and AIMer-192s affect fan-out, leaf page usage, insertion throughput, and storage cost under identical experimental conditions.
3. We design and evaluate a storage-engine-aware mitigation architecture that combines a split-table schema with Merkle Tree-based batch signing, so that document rows retain only fixed-size hashes, proof paths, and batch identifiers while full PQC signatures are stored once per batch.

The rest of this paper is organized as follows. Section 2 provides theoretical background on the InnoDB B^+-Tree storage structure, the off-page overflow model, Merkle Tree-based batch integrity verification, and the four signature algorithms used in the experiments. Section 3 surveys related work on PQC signature performance, database integrity mechanisms, and large object storage in relational databases. Section 4 describes the hardware and software environment, the metrics collected, and the four schema configurations under comparison. Section 5 presents the experimental results: first, the B^+-Tree degradation observed in the single-table baseline, and then the evaluation of the split schema, Merkle batch signing, and the combined architecture. Section 6 concludes with a summary of findings, algorithm-specific recommendations, and directions for future work.

2 Background

This section covers three theoretical backgrounds for our experiments. We first examine MySQL InnoDB's storage structure and the B^+-Tree index mechanism, then analyze how large signatures affect off-page storage. We then present the theoretical basis for Merkle Tree-based batch signing, which is central to our mitigation architecture, and finally compare the parameters of the four signature algorithms used in the experiments.

2.1 MySQL InnoDB and B^+-Tree Index Structure

MySQL InnoDB is the most widely used Relational Database Management System (RDBMS) storage engine for applications that require transactional integrity [12]. InnoDB supports Atomicity/Consistency/Isolation/Durability (ACID) transactions, row-level locking, and write-ahead redo logs, and stores all table data in a B^+-Tree as a *clustered index* indexed by the primary key. Unlike secondary indexes, the clustered index stores the full row in each leaf node, so a single primary key lookup retrieves the complete record.

The B$^+$-Tree divides the sorted key space into a hierarchy of pages. Internal pages store key-pointer pairs that guide traversal, while leaf pages store the actual row data. For a table with n rows and a leaf fan-out f, the number of leaf pages is $\lceil n/f \rceil$, and the tree depth is approximately $\lceil \log_f n \rceil$. Since InnoDB uses a fixed default page size of 16 KB, the leaf fan-out is determined as follows.

$$f = \left\lfloor \frac{P_{\text{page}}}{r_{\text{avg}}} \right\rfloor, \quad P_{\text{page}} = 16{,}384\,\text{B} \tag{1}$$

Here, r_{avg} is the average row size including InnoDB row overhead. As fan-out decreases, tree depth increases, causing additional page I/O per point lookup and reducing lookup and insertion efficiency. Therefore, the starting point of this study is that storing large signatures inline in a row causes r_{avg} to grow, leading to fan-out collapse.

2.2 Off-page Storage and Theoretical Row Size Model

The InnoDB DYNAMIC row format, the default since MySQL 5.7, decides off-page storage based on the rule that at least two rows must fit in a page [11]. Specifically, if the total row size due to inline storage of variable-length columns exceeds half of the page's usable space (approximately 8,000 B), the column data is moved to a dedicated overflow page, leaving only a 20-byte pointer in the leaf page inline area.

In this MySQL manual model, the effective inline row size for a signature field of s_{sig} bytes is as follows.

$$r_{\text{theory}} = \begin{cases} r_{\text{fixed}} + s_{\text{sig}} & \text{if } s_{\text{sig}} \le 8{,}000\,\text{B}, \\ r_{\text{fixed}} + 20 & \text{if } s_{\text{sig}} > 8{,}000\,\text{B}. \end{cases} \tag{2}$$

Here, r_{fixed} is the sum of fixed-size columns (excluding the variable-length signature column) and the InnoDB row overhead. Note that this model is a purely theoretical prediction. The actual off-page trigger condition in InnoDB depends not on column size alone, but on whether the entire row fits in the page. Therefore, in Section 5, we analyze the specific discrepancy between theoretical and measured values.

2.3 Merkle Tree-Based Batch Integrity Verification

The Merkle Tree is a cryptographic data structure proposed by Ralph Merkle in 1979 and formally published in 1987 [1][8], designed for efficient integrity verification of large datasets. Each leaf node stores the hash of an individual data block, and parent nodes are computed by combining the hashes of their two children. The final single root hash summarizes the integrity of the entire dataset, and provides an *inclusion proof* for any leaf node using only $O(\log N)$ hash operations.

In this study, we apply this structure to mitigate PQC signature overhead. By grouping B documents into one batch, computing the Merkle root, and signing the root once with a PQC signature, the integrity of all B documents is guaranteed. Figure 1 shows the Merkle Tree structure and inclusion proof path using a simplified example with $B = 4$. The dashed nodes form the proof_path for verifying. Signing the root once with a PQC signature guarantees the integrity of the entire batch, and verifying an individual document requires only $O(\log B)$ SHA-256 operations.

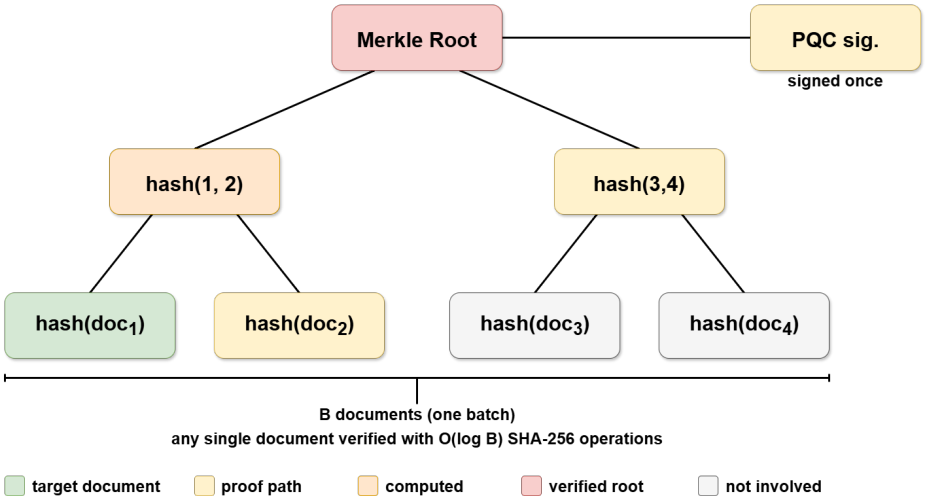

Fig. 1. Merkle Tree structure (B=4 example)

2.4 Signature Algorithms and Parameters

This study compares four algorithms at the same security level. **RSA-7680** is a classical algorithm based on the integer factorization problem, producing a 960-byte signature. **CRYSTALS-Dilithium (ML-DSA)** is based on the hardness of the Module Learning With Errors (MLWE) and Module Short Integer Solution (MSIS) problems and is known to be resistant to quantum attacks [3]. We use Dilithium3 (ML-DSA-65, 3,293 bytes) in this study. **AIMer** is a PQC signature scheme based on the MPC-in-the-head (MPCitH) paradigm, whose security relies on the one-wayness of a symmetric-key primitive (AIM), constructed as a signature scheme via the Fiat-Shamir transform [2]. We include both the fast variant (AIMer-192f) and the short variant (AIMer-192s).

Table 1. Comparison of signature algorithm parameters used in the experiments

Algorithm	Cryptographic basis	Signature size (B)	Security level	Standard
RSA-7680	Integer factorization	960	∼192-bit (classical)	Classical
Dilithium3	Lattice	3,293	NIST Level 3	FIPS 204
AIMer-192f	MPCitH	13,056	NIST Level 3	KpqC
AIMer-192s	MPCitH	9,120	NIST Level 3	KpqC

3 Related Work

This study sits at the intersection of three research streams: first, performance overhead of PQC signatures; second, integrity mechanisms in relational databases; and third, large object storage in RDBMS.

3.1 Post-Quantum Cryptographic Signature Performance

The computational and communication overhead of PQC signatures has been actively studied across various deployment environments. Sikeridis et al. benchmarked several NIST candidate algorithms within TLS 1.3 handshakes, showing that large signature and public key sizes increase handshake latency and packet fragmentation, especially on high-latency links [17]. Mallick et al. analyzed PQC integration in blockchain systems, finding that Dilithium signatures increase per-transaction size by 3–4x compared to ECDSA, with a corresponding impact on block propagation time [7]. These studies consistently identify signature size as the primary overhead factor, but focus on network transmission and computation latency. The structural impact on B^+-Tree indexes when PQC signatures are stored as persistent records in an RDBMS has not been studied.

3.2 Database Integrity Mechanisms and Merkle Structures

Several mechanisms have been proposed to guarantee integrity in relational databases. Hash-chain logging, where each record stores a hash of the previous record, is widely used in audit-trail systems and append-only ledgers [15]. This approach adds only 32 bytes per row, but has the limitation of not providing cryptographic commitment to external parties.

Merkle Trees enable efficient membership proofs and batch integrity verification. This structure underpins transaction integrity in Bitcoin [9] and Certificate Transparency logs [5], supporting compact proofs that a given item is included in a log. However, in these applications the underlying storage systems for Merkle Trees are purpose-built key-value or log-structured stores.

3.3 Large Object Storage in Relational Databases

Managing large variable-length fields within B^+-Tree storage engines has been studied in the context of BLOB and TEXT columns. PostgreSQL adopts the TOAST mechanism, which compresses and splits large values into a separate storage table [13]. These mechanisms are designed to handle large values easily, but were engineered under the assumption that large fields are exceptional. In a PQC environment where every row in a table contains a signature of several kilobytes, the off-page mechanism is invoked for all rows, fundamentally changing index access patterns. The performance implications of this universal large-field scenario have not been quantified for PQC signature sizes.

4 Experimental Setup and Methodology

This section describes the hardware and software environment, the metrics collected, and the schema configurations under comparison. All experiments were conducted under controlled conditions on the same platform.

4.1 Hardware and Software Environment

The experiments were performed on an Apple M2 MacBook Air with 8 GB of unified memory. Tables 2 and 3 summarize the hardware and software specifications. The database server is MySQL 9.6.0, with InnoDB configured with a 128 MB buffer pool, 16 KB page size, and DYNAMIC row format. The benchmark program is implemented in Python 3.14.2, using the `pymysql` library for database connectivity, the `dilithium-py` library for Dilithium3 signing operations, and a custom AIMer library for AIMer-192f/s signing operations.

Table 2. Hardware Environment

Item	Details
System	Apple M2 MacBook Air (Mac14,2)
Processor	Apple M2 (4 performance cores + 4 efficiency cores, 8 cores total)
Memory	8 GB unified memory
Storage	Built-in NVMe SSD

4.2 Metrics and Schema Configurations

The following metrics are collected for each experimental configuration. **B^+-Tree fan-out** is estimated using Equation (1), where r_{avg} is measured by dividing the table data size (queried from MySQL internal statistics tables) by the number of rows. **Leaf page count** is collected from MySQL internal statistics tables. **Insertion throughput** is measured as the total time to insert 10^6

Table 3. Software Environment

Item	Details
Operating System	macOS 26.3.1
DBMS	MySQL 9.6.0
Storage Engine	InnoDB
Row Format	DYNAMIC
Durability Setting	Flush to disk on each transaction commit
Buffer Pool	128 MB
Page Size	16 KB
Benchmark Language	Python 3.14.2
DB Connection Library	`pymysql`
Dilithium3 Library	`dilithium-py`
AIMer Library	Custom AIMer wrapper

rows under the following conditions. For the single-table and split-only configurations, signatures are pre-generated before the insertion loop and the same bytes are reused for all rows, so the measured time reflects pure DB insertion overhead. For the combined architecture, the Merkle Tree is constructed in real time and one PQC signature is computed per batch of 512 documents, so the measured time includes both DB insertion and batch-level cryptographic operations. This design makes the comparison conservative: the throughput advantage of the combined architecture reported in Section 5 is a lower bound, since real per-document signing would make the single-table baseline substantially slower.

Table 4. Schema Configurations Under Comparison

Configuration	Description
Single-table	The full signature is stored inline in a single table. Classical baseline.
Split-only	The signature is moved to a separate signature table; one full signature is kept per document.
Merkle without split	Merkle batch signing ($B = 512$) is applied, but the proof path and the full signature are stored in the same document row.
Combined	A combination of the split schema and Merkle batch signing. The document table stores only the proof path; the signature is stored once per batch in a separate table.

Figure 2 visually compares the table structures and key differences of the four configurations.

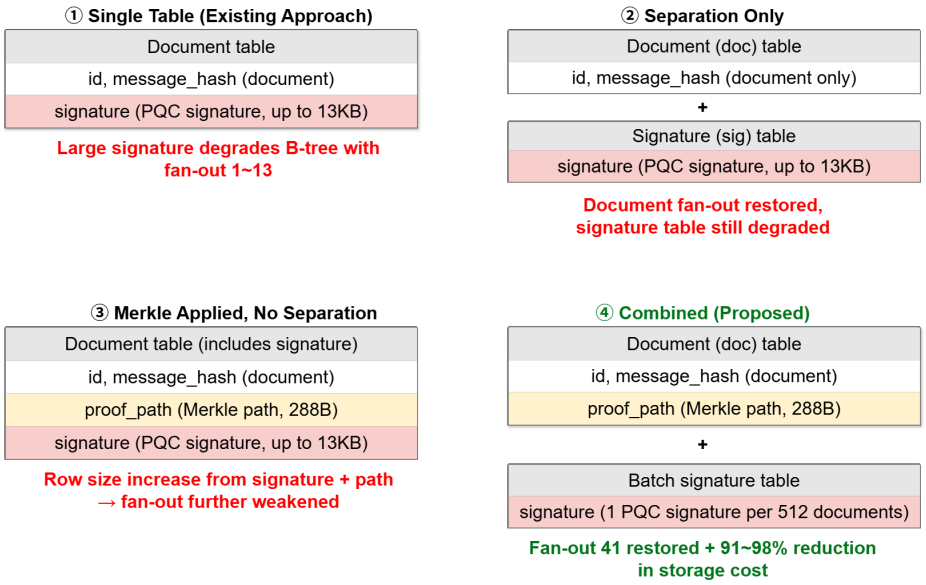

Fig. 2. Comparison of table structures for the four schema configurations.

5 Experimental Results

This section is organized in two parts. First, we quantify the effect of PQC signatures on the InnoDB B$^+$-Tree structure using a single-table benchmark. Then, we compare three mitigation strategies: the split schema alone, Merkle batch signing alone, and the combined architecture.

5.1 Single-Table Baseline: B$^+$-Tree Degradation

In the baseline experiment, we evaluate the simplest approach of storing the full signature inline in a single table. We insert one million rows for each algorithm and collect B$^+$-Tree metrics.

Listing 1.1. Single-table baseline schema

```
CREATE TABLE bench_{algo} (
    id           BIGINT AUTO_INCREMENT PRIMARY KEY,
    message_hash BINARY(32), -- Document hash (32 bytes, fixed)
    signature    BLOB          -- Full PQC signature
);
```

As shown in Listing 1.1, `message_hash` is a 32-byte hash that identifies the document, and `signature` stores the full PQC signature, whose size varies by algorithm. To measure the impact of signature size on InnoDB pages (16 KB), no unnecessary columns other than the signature are included.

The theoretical predictions using Equation (2) are as follows.

Table 5. Theoretical Row Size and Fan-out Predictions

Algorithm	Signature (B)	Off-page	Theoretical row size (B)	Theoretical fan-out
RSA-7680	960	No	$78 + 960 = 1{,}038$	≈ 15
Dilithium3	3,293	No	$78 + 3{,}293 = 3{,}371$	≈ 4
AIMer-192s	9,120	Yes	$78 + 20 = 98$	≈ 167
AIMer-192f	13,056	Yes	$78 + 20 = 98$	≈ 167

RSA-7680 has a row size of $r = 78 + 960 = 1{,}038$ B and a predicted fan-out of ≈ 15, while Dilithium3 has $r = 3{,}371$ B and a predicted fan-out of ≈ 4. For AIMer-192f and AIMer-192s, which exceed the 8,000 B threshold, off-page storage is triggered, giving an effective row size of $r = 78 + 20 = 98$ B and a predicted fan-out of ≈ 167. However, the measured values shown in Table 6 differ significantly from these predictions.

Table 6. Single-Table Benchmark Results (1,000,000 rows)

Algorithm	Signature (B)	Theoretical fan-out	Actual fan-out	Leaf pages	Insert time (s)
RSA-7680	960	15	13	71,438	89.9
Dilithium3	3,293	4	3	250,031	342.3
AIMer-192s	9,120	167	1	1,005,499	1,087.9
AIMer-192f	13,056	167	1	1,005,499	1,452.5

The most important finding appears in the AIMer algorithms. Theoretically, a fan-out of ≈ 167 was predicted for both AIMer variants based on exceeding the 8,000 B threshold. However, the measured fan-out is 1 for both, meaning exactly one row exists per leaf page. This discrepancy occurs because InnoDB did not trigger off-page storage in this case. The full row size for AIMer-192f is $r = 78 + 13{,}056 = 13{,}134$ B, which fits within a single page (16,384 B). However, a second row (26,268 B) would exceed the page capacity, so each page holds exactly one row. This effectively turns the B^+-Tree into a linear structure. As a result, the insertion time increases to 1,452.5 s, which is 16.2x compared to RSA-7680. This shows that the traditional single-table storage approach is not suitable for large PQC signatures.

Figure 3 visually compares the B$^+$-Tree leaf structures of the two algorithms.

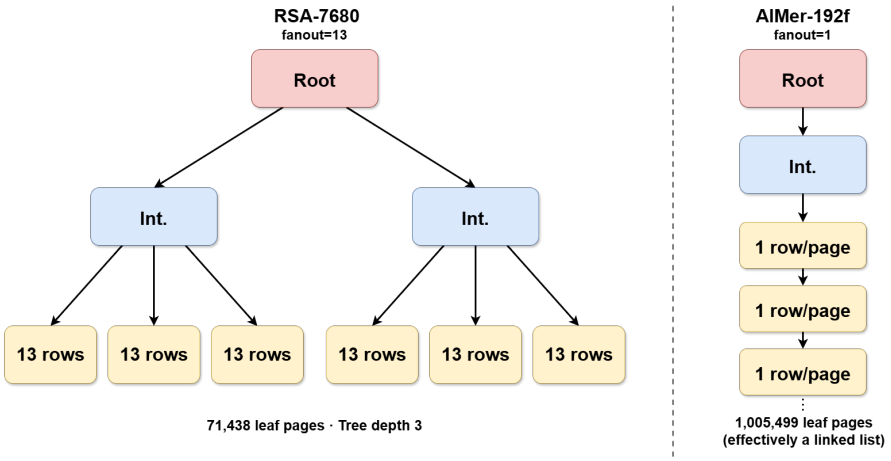

Fig. 3. B$^+$-Tree leaf structure comparison. RSA-7680 (left), AIMer-192f (right)

5.2 Combined Mitigation Architecture

The results from Section 5.1 show that the root cause of B$^+$-Tree degradation is the coexistence of large signatures and document records in the same row. To address this, we evaluate two candidate mechanisms—the split-table schema and Merkle batch signing—each independently and in combination.

5.2.1 Split Schema Only

The split schema moves signatures to a dedicated `sig` table, leaving only the document hash in the document table. As shown in Table 7, the document table fully restores the B$^+$-Tree regardless of the algorithm. However, since the signature table retains one million full signatures, the same fan-out collapse as in the single-table baseline persists. In the case of AIMer-192f, 15,748 MB is still required even under the split configuration, with no reduction in storage space. The split schema resolves the document index problem but does not reduce the volume of signature data. In Table 7, row (B) refers to the average space allocated per row. Since both AIMer sig table variants have fan-out = 1, a full page is allocated per row, resulting in the same value for both. The actual row sizes are 13,110 B for AIMer-192f and 9,174 B for AIMer-192s.

5.2.2 Merkle Batch Signing Only

This experiment is designed to verify whether applying Merkle batch signing without a split schema improves the B$^+$-Tree structure. In Merkle batch signing,

Table 7. Split Schema Only: B$^+$-Tree Metrics for Document and Signature Tables

	Document Table			Signature Table		
Algorithm	**Row (B)**	**Fan-out**	**Leaf pages**	**Row (B)**	**Fan-out**	**Leaf pages**
RSA-7680	65	253	3,923	1,095	14	66,676
Dilithium3	65	253	3,923	4,102	3	250,031
AIMer-192s	65	253	3,923	16,449	1	1,003,909
AIMer-192f	65	253	3,923	16,449	1	1,003,909

each document must store a proof path to prove that it belongs to the signed batch. The proof path consists of the sibling node hashes along the path from the document's hash to the root in the Merkle Tree, and has a size of $9 \times 32 = 288$ B when $B = 512$.

Without a split schema, this proof path (288 B) and the full PQC signature are stored together in the same document row. Since only the proof path is added without changing where the signature is stored, the row size actually increases.

As shown in Table 8, the fan-out for Dilithium3 remains unchanged at 3. For AIMer-192f, the situation gets worse: with the 288 B proof path added, the allocated space per row reaches 16,788 B. Since there is not enough page capacity to hold a second row, the fan-out remains at 1. In fact, as the row grows larger, page splits occur more frequently during insertion, increasing the number of leaf pages to 1,024,515—more than in the single-table baseline. This experimentally shows that Merkle batch signing alone provides no improvement to the B$^+$-Tree structure.

Table 8. Merkle Batch Signing Without Split Schema ($B = 512$): B$^+$-Tree Metrics

Algorithm	**Row size (B)**	**Fan-out**	**Leaf pages**	**Data (MB)**
Dilithium3	4,102	3	250,029	3,912.0
AIMer-192s	16,788	1	1,024,515	16,009.9
AIMer-192f	16,788	1	1,024,515	16,009.9

The two mechanisms are therefore complementary. The split schema restores fan-out by removing signatures from document rows, but does not solve the storage problem. Merkle batch signing reduces storage by amortizing the signature cost, but without a split schema it maintains or worsens B$^+$-Tree degradation. Only when the split schema ensures that document rows contain only the proof path, and the Merkle mechanism ensures that signatures are shared across B documents (in this study, $B = 512$), are both problems solved simultaneously.

5.2.3 Combined Architecture

The combined architecture consists of two table types. The document table stores one row per document and includes only the proof path instead of the signature. The batch signature table stores one row per 512 documents, holding the Merkle root and PQC signature.

```
CREATE TABLE integrity_merkle_b512 (
    id           BIGINT AUTO_INCREMENT PRIMARY KEY,
    message_hash BINARY(32)  NOT NULL,
    proof_path   VARBINARY(288) NOT NULL,
    batch_id     BIGINT       NOT NULL,
    INDEX idx_batch (batch_id)
);
```

Listing 1.2. Combined architecture schema: document table

```
CREATE TABLE integrity_batch_{algo}_b512 (
    id          BIGINT AUTO_INCREMENT PRIMARY KEY,
    merkle_root BINARY(32) NOT NULL,
    signature   BLOB      NOT NULL
);
```

Listing 1.3. Combined architecture schema: batch signature table

At insertion time, a Merkle Tree is built from 512 document hashes, and the root is signed once with a PQC signature. Each document stores only its proof path, and at retrieval time the batch root is fetched via a JOIN and the proof is verified with 9 SHA-256 operations. Applying the combined architecture restores the document table fan-out to 41, and reduces the number of leaf pages from 1,005,499 (AIMer single-table) to 23,813—a reduction of over 97%. Since the document table contains no algorithm-dependent data, this figure is the same for all PQC algorithms.

Insertion Throughput As shown in Table 9, insertion throughput improves dramatically. The combined architecture achieves 19,341 rows/s, which is 28.1× higher than the AIMer-192f single-table configuration (688 rows/s). The throughput is identical across all algorithms, meaning that in the combined architecture the insertion load no longer depends on signature size.

Table 9. Insertion Throughput: Single-table vs. Combined Architecture ($B = 512$)

Schema	Algorithm	Throughput (rows/s)	Time (s)
Single-table	RSA-7680	11,123	89.9
	Dilithium3	2,921	342.3
	AIMer-192f	688	1,452.5
	AIMer-192s	919	1,087.9
*Combined ($B = 512$)	All algorithms	19,341	51.7

Amortized Storage Cost The per-document signature storage cost decreases by 91–98% at $B = 512$ (Table 10).

Table 10. Amortized Storage Cost per Document ($B = 512$)

Algorithm	Original signature (B)	*Combined (B)
Dilithium3	3,293	294.4 (−91.1%)
AIMer-192f	13,056	313.5 (−97.6%)
AIMer-192s	9,120	305.8 (−96.6%)

Figure 4 compares the per-document storage cost of the single-table and combined configurations on a log scale.

Table 11 provides a comprehensive comparison of the four schema configurations. The combined architecture is the only configuration that simultaneously achieves B^+-Tree fan-out recovery, reduction in leaf page count, improved insertion throughput, and reduced per-document storage cost across all algorithms.

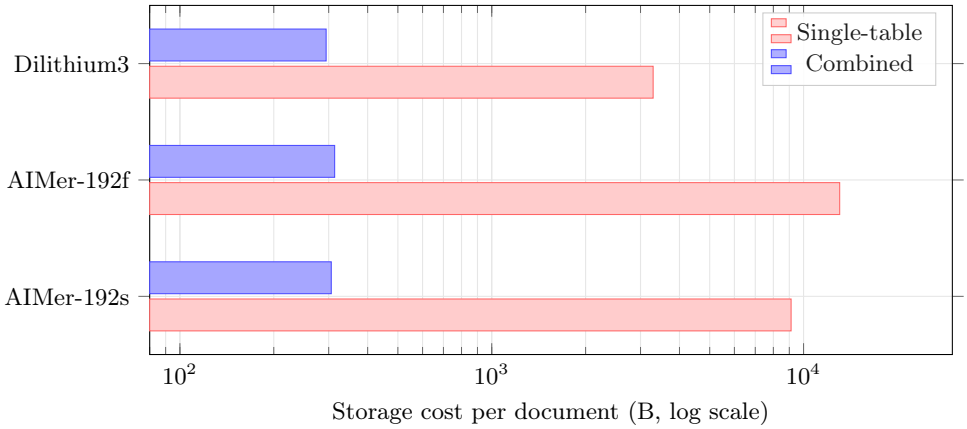

Fig. 4. Comparison of per-document signature storage cost (log scale). The combined architecture ($B = 512$) reduces storage cost by -91.1% for Dilithium3, -97.6% for AIMer-192f, and -96.6% for AIMer-192s.

Table 11. Summary Comparison by Configuration ($B = 512$)

Algorithm	Metric	Single -table	Split -only	Merkle (no split)	*Combined
Dilithium3	Doc fan-out	3	253	3	**41**
	Doc leaf pages	250,031	3,923	250,029	**23,813**
	Insert time (s)	342.3	379.1	338.3	**51.7**
	Storage/doc (B)	3,293	3,293	3,293+288	**294.4**
AIMer-192f	Doc fan-out	1	253	1	**41**
	Doc leaf pages	1,005,499	3,923	1,024,515	**23,813**
	Insert time (s)	1,452.5	1,578.0	1,187.1	**51.7**
	Storage/doc (B)	13,056	13,056	13,056+288	**313.5**
AIMer-192s	Doc fan-out	1	253	1	**41**
	Doc leaf pages	1,005,499	3,923	1,024,515	**23,813**
	Insert time (s)	1,087.9	1,201.0	880.0	**51.7**
	Storage/doc (B)	9,120	9,120	9,120+288	**305.8**

6 Conclusion

Post-quantum cryptographic signatures cause severe and algorithm-dependent degradation. Dilithium3 reduces the B$^+$-Tree fan-out from 13 (RSA-7680) to 3, while AIMer-192f and AIMer-192s collapse the fan-out to 1, turning the struc-

ture entirely linear. Unlike the theoretical model, which predicted a fan-out of approximately 167 based on off-page storage, AIMer signatures are stored inline, resulting in exactly one row per leaf page. Insertion time increases by up to 16×. Therefore, the traditional single-table storage approach is not suitable for post-quantum cryptographic algorithms with large signatures.

With a batch size of $B = 512$, the combined architecture restores the B^+-Tree fan-out to 41, reduces per-document signature storage by 91–98%, and improves insertion throughput by up to 28.1× compared to AIMer. Integrity is maintained through a per-document Merkle proof path, which enables cryptographic verification without storing a full PQC signature per document.

The experimental results suggest the following recommendations by algorithm:

Dilithium3: Fan-out drops from 13 to 3 in the single-table configuration. The combined architecture restores the fan-out to 41 and significantly reduces insertion time.

AIMer-192f and AIMer-192s: In the single-table case, the fan-out collapses to 1, rendering the B^+-Tree non-functional as an index structure. The combined architecture restores the fan-out to 41, reduces per-document signature storage by 96–98%, and $B = 512$ is the recommended batch size, offering excellent insertion throughput.

The proposed architecture is best suited for *append-heavy* workloads where documents are written once and read many times. In such scenarios, updates are rare compared to reads, so the write amplification caused by rewriting B proof paths per document update is negligible.

However, the following limitations should be considered when interpreting the results of this study. First, the current implementation measures only insertion throughput. A quantitative benchmark of the cost of updating a single document, including proof path regeneration across B rows, is left for future work. Second, end-to-end latency including real-time per-document signing for the single-table and split-only configurations has not been measured; the throughput advantage of the combined architecture reported in Section 5 is therefore a conservative lower bound.

Appendix

Abbreviation	Full Name
ACID	Atomicity, Consistency, Isolation, Durability
AIM	Advanced Invertible Map
BLOB	Binary Large Object
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
IoT	Internet of Things
KpqC	Korea Post-Quantum Cryptography
ML-DSA	Module-Lattice-Based Digital Signature Algorithm
MLWE	Module Learning With Errors
MPCitH	Multi-Party Computation in the Head
MSIS	Module Short Integer Solution
NIST	National Institute of Standards and Technology
NVMe	Non-Volatile Memory Express
PQC	Post-Quantum Cryptography
RDBMS	Relational Database Management System
RSA	Rivest–Shamir–Adleman
SHA	Secure Hash Algorithm
SSD	Solid-State Drive
TLS	Transport Layer Security
TOAST	The Oversized-Attribute Storage Technique

Table 12. Abbreviations used in this paper.

References

1. Aki: Digital signatures: a tutorial survey. Computer **16**(2), 15–24 (1983)
2. Cho, Jihoon and Ha, Jincheol and Kim, Seongkwang and Kwon, Jihoon and Lee, Byeonghak and Lee, Joohee and Lee, Jooyoung and Lee, Sangyub and Moon, Dukjae and Son, Mincheol and others: The AIMer Signature Scheme
3. Dang, Thinh and Lichtinger, Jacob and Liu, Yi-Kai and Miller, Carl and Moody, Dustin and Peralta, Rene and Perlner, Ray and Robinson, Angela and others: Module-lattice-based digital signature standard (2024)
4. Korea Post-Quantum Cryptography Research Group (KpqC): Announcement on the Final Algorithms (4 Types) of the National Post-Quantum Cryptography Competition. `https://kpqc.or.kr/contents/03_exhibit/sub_03.html` (January 2025), Accessed: 2026-05-05
5. Laurie, B., Messeri, E., Stradling, R.: RFC 9162: Certificate Transparency Version 2.0 (2021)

6. Lee, S.W., Kim, H.G., Song, M.H., Eum, S.W., Seo, H.J.: Accelerating FAEST signatures on ARM: NEON SIMD AES and parallel VOLE optimization. Cryptology ePrint Archive (2026)
7. Mallick, T., Zeldin, M., Cenk, M., Nita-Rotaru, C.: Quantum Disruption: An SOK of How Post-Quantum Attackers Reshape Blockchain Security and Performance. arXiv preprint arXiv:2512.13333 (2025)
8. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the theory and application of cryptographic techniques. pp. 369–378. Springer (1987)
9. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
10. National Institute of Standards and Technology (NIST): NIST Releases First 3 Finalized Post-Quantum Encryption Standards. https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards (August 2024), Accessed: 2026-05-05
11. Oracle Corporation: MySQL 5.7 Reference Manual: 14.11 InnoDB Row Formats - DYNAMIC Row Format. Oracle (2023), https://dev.mysql.com/doc/refman/5.7/en/innodb-row-format.html#innodb-row-format-dynamic
12. Oracle Corporation: MySQL 9.7 Reference Manual. Oracle (2025), https://dev.mysql.com/doc/refman/9.7/en/
13. PostgreSQL Global Development Group: PostgreSQL 16.0 Documentation: Chapter 73. TOAST (2023), https://www.postgresql.org/docs/current/storage-toast.html, Accessed: 2026-05-05
14. Raavi, M., Khan, Q., Wuthier, S., Chandramouli, P., Balytskyi, Y., Chang, S.Y.: Security and performance analyses of post-quantum digital signature algorithms and their TLS and PKI integrations. Cryptography **9**(2), 38 (2025)
15. Schneier, B., Kelsey, J.: Secure audit logs to support computer forensics. ACM Transactions on Information and System Security (TISSEC) **2**(2), 159–176 (1999)
16. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Review **41**(2), 303–332 (1999). https://doi.org/10.1137/S0036144598347011, https://doi.org/10.1137/S0036144598347011
17. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-quantum authentication in TLS 1.3: A performance study. Cryptology ePrint Archive (2020)
18. Thenmozhi, K., Begum, R.S.: Blockchain-Integrated Overhead-Aware Optimization of Post-Quantum Digital Signature Protocols. In: 2025 International Conference on Electrical Engineering and Informatics (ICEEI). pp. 1–7. IEEE (2025)