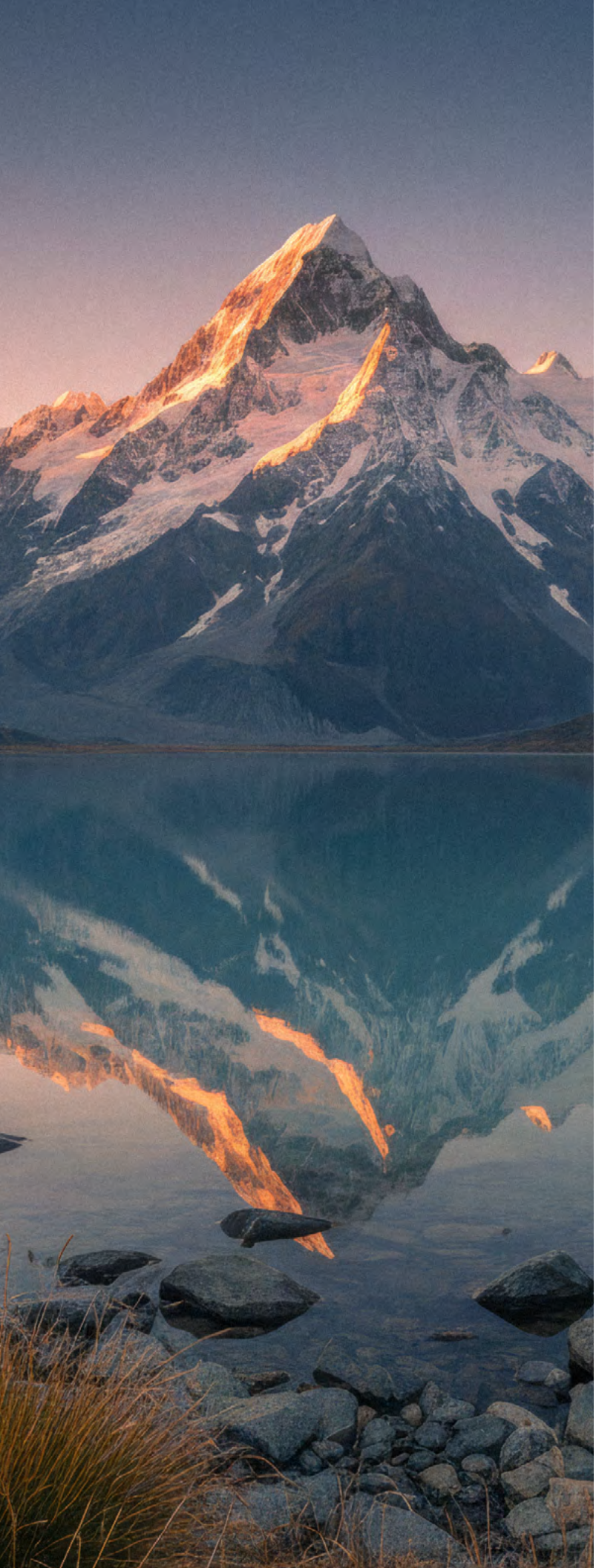
PERCONA

Whitepaper

THE PERFORMANCE COST OF TRANSPARENT DATA ENCRYPTION IN POSTGRESQL

A benchmark-backed look at pg_tde 2.2.2, including the numbers most vendors don't show you



Introduction

What Is the Real Performance Impact of Transparent Data Encryption in PostgreSQL?

Every team evaluating encryption at rest for PostgreSQL asks the same question: what will this cost us in performance? Most vendors answer with a single benchmark graph and move on. That graph is usually true and usually misleading, because standard tests never touch the code paths where encryption actually costs something. One graph cannot tell the full story.

Teams adopting transparent data encryption (TDE) are usually doing it under pressure: an auditor citing GDPR, HIPAA, PCI-DSS, or SOX; a customer contract requiring encryption at rest; a security review that flagged unencrypted data files as a finding that has to close by a specific date. Under that kind of pressure, the easy path is to grab a single benchmark, run a quick sysbench pass, and call it done. That's exactly the problem. A quick benchmark shows near-zero overhead for almost any TDE implementation, because it never generates the sustained write volume or the WAL contention where encryption cost actually appears. Vendors that publish only that graph aren't lying. They're just not showing you the whole picture.

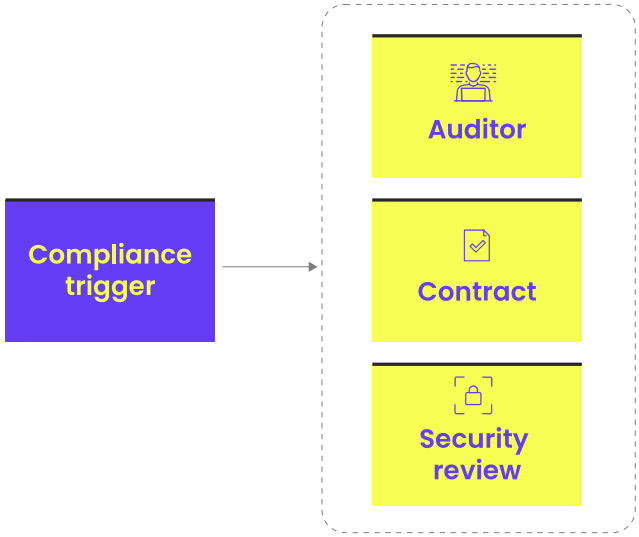
This whitepaper answers the question properly, using reproducible benchmarks on hardware chosen to expose overhead rather than hide it. For most workloads, relation file encryption has little to no measurable impact. Write-heavy operations and high WAL churn can expose more significant overhead, and we name exactly where. The short version:

- **Data file encryption** with pg_tde stays at or below 2% for typical workloads, inside the noise of production-like testing.
- **Realistic OLTP read-write workloads,** including WAL encryption, show only a few percent difference end to end, under 10% in every configuration tested.
- **The measurable cost** shows up only in specific, identifiable corners: bulk table rewrites (10-30%) and sustained, high-concurrency WAL volume far beyond typical production traffic (20%, down from over 40% before pg_tde 2.2.2).

This whitepaper also covers the improvements shipped in pg_tde 2.2.2 and the architectural work already underway to close the remaining WAL-encryption gap further. It's built on benchmark work from the pg_tde engineering team, condensed for technical and business readers who need a straight technical breakdown before they commit to an encryption strategy, and who want to know exactly where the cost lives rather than whether one exists at all.

We're publishing the worst cases alongside the good ones, with the hardware and methodology behind them, because a benchmark you can't reproduce isn't evidence. If you're deciding whether encryption at rest belongs in your PostgreSQL environment, this is the data to decide on.

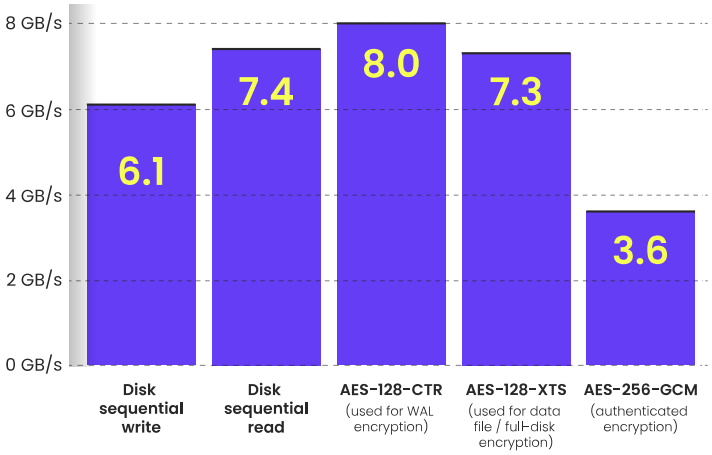
Common Reasons to Adopt TDE



Why Encryption Is Nearly Free on Modern Hardware

Start with the physics. Encryption is genuinely CPU work, and on older hardware it showed up in every benchmark that touched it. That's no longer true. Modern CPUs ship AES-NI, a dedicated instruction set that accelerates AES encryption directly in hardware, and the throughput numbers make the case on their own.

Single-core benchmarks on a 32-core AMD Threadripper 3970X with an Intel Optane P5800X SSD show:



A single core encrypting with AES-128-CTR moves data faster than this test rig's disk can write it. That's the core insight: on hardware built in the last several years, encryption isn't the bottleneck in most database workloads. Disk and memory bandwidth usually are. When a workload is already limited by disk or memory rather than CPU, encryption can ride on CPU cycles that were sitting idle anyway, at little or no additional cost.

This mirrors something most infrastructure teams already trust without a second thought: full-disk encryption. LUKS and BitLocker apply the same class of AES acceleration to every byte written to disk, across nearly every laptop and server fleet in production today, and nobody budgets for a performance hit when they turn it on. Database-level TDE isn't a fundamentally different technology. It's the same acceleration, applied one layer up the stack, at the page level instead of the block-device level.

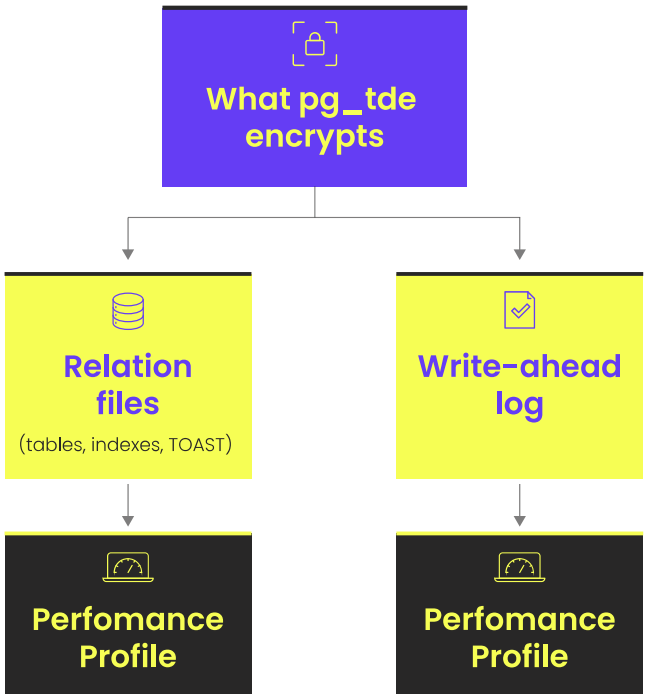
The real question isn't whether encryption is fast in isolation. It's whether it's integrated well into PostgreSQL's actual read and write paths, and that depends on which files are being encrypted.

What pg_tde Actually Encrypts

"How fast is TDE?" doesn't have one answer, because encrypting a PostgreSQL database isn't one operation. pg_tde encrypts two different things, and they behave completely differently under load:

- **Relation files** — the actual database tables, indexes, and TOAST data stored on disk
- **The write-ahead log (WAL)** — the record of every change, written before it's applied to the database, used for crash recovery and replication

These two behave almost like opposites. Relation file encryption is close to free in nearly every realistic scenario, for reasons explained in the next section. WAL encryption is where real overhead can show up, because of how PostgreSQL's WAL is built. A single benchmark graph usually reports one blended number for "TDE overhead," which hides both the good news (relation files cost almost nothing) and the part that actually needs explaining (WAL, under specific conditions).



Relation File Encryption: The Overhead You Won't Notice

For the overwhelming majority of PostgreSQL workloads, relation file encryption is not measurable in any meaningful way. Typical sysbench-style OLTP workloads and simple sequential reads show 2% overhead or less, a number that sits inside the normal noise of benchmarking on real hardware. Isolating a true 1-2% difference requires carefully controlled lab conditions; it doesn't show up as a distinguishable cost in a production environment.

The mechanism is straightforward. PostgreSQL reads and writes relation data in fixed 8kB blocks, and pg_tde encrypts and decrypts at that same block boundary, before a write to disk and after a read from disk. Once a page is in shared buffers, it stays decrypted in memory. Read-heavy and cache-friendly workloads never touch the encryption path on the hot path at all. Writes mostly happen through the checkpointer and background writer, not in the client-facing backend process, so encryption doesn't add latency to the query a user is waiting on. This is the same behavior full-disk encryption exhibits, and it's why that comparison holds up under scrutiny rather than just being a convenient analogy.

There is a real exception, and it's worth stating plainly rather than burying it: single-threaded, write-heavy operations that rewrite large volumes of table data all at once.

Operation	Worst-Case Overhead	Why
`CREATE TABLE AS SELECT`	Up to 30%	Reads, decrypts, re-encrypts, and writes every page with no other work to hide the cost behind
`ALTER TABLE` (full rewrite)	Up to 15%	Same read/decrypt/encrypt/write pattern, plus schema work
`VACUUM FULL`	Up to 10%	Rewrites the entire table into a new file
`UPDATE` touching all/most rows	Up to 10%	Includes checkpoint time, where most of the actual write work happens

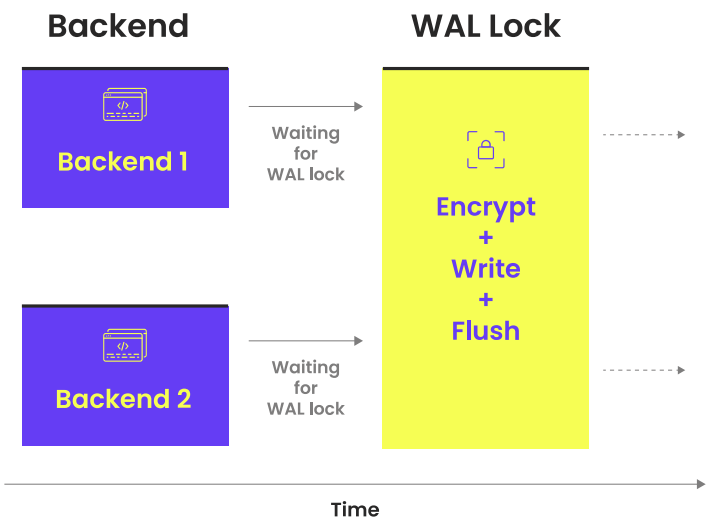
The mechanism behind the CTAS number is simple: an operation that copies a table with no other work to do is normally limited only by disk speed. Add encryption, and every page has to be decrypted on read and re-encrypted on write, roughly halving the effective throughput from around 8 GB/s to 4 GB/s of AES-128-CTR bandwidth. On fast enough storage, that turns a disk-bound operation into an encryption-bound one. These numbers also require storage that can sustain very high write speeds continuously; most cloud volumes and consumer SSDs can only hit those speeds in short bursts, so the practical impact on typical infrastructure is smaller than the worst-case lab number suggests.

The honest framing: these are maintenance operations, not steady-state application traffic. If your workload runs occasional `VACUUM FULL` passes or nightly `CREATE TABLE AS SELECT` jobs, budget for a slower maintenance window. If your workload is standard OLTP traffic, this section doesn't apply to you at all.

WAL Encryption: Where the Real Engineering Challenge Lives

This is the part most TDE vendors don't explain, because it's harder to explain and less flattering to show. It's also the part that matters most if your workload does sustained, high-concurrency writes.

PostgreSQL allows only one process to flush the WAL at a time. Every backend that wants to commit a change has to acquire a single WAL lock, write and flush its data while holding it, then release it for the next backend in line. This is a serialization point by design; it's what makes crash recovery reliable. Most WAL encryption implementations, pg_tde's earlier releases and the current version included, encrypt the data while that lock is held. That means every commit now pays the cost of encryption during the one window where every other backend is already waiting.



Under normal OLTP traffic, this cost is small: under 10% end to end in every configuration tested, consistent with the "few percent" difference typical read-write benchmarks show. It only becomes significant under a specific, extreme condition: many parallel sessions generating a sustained, very high volume of WAL, well beyond what typical production traffic produces. In benchmarks built specifically to stress that scenario (32 cores, over 1 GB/s of sustained WAL, many concurrent sessions), pg_tde's WAL encryption overhead measured at roughly 40%+ before version 2.2.2.

That is a real number, and we're not going to pretend otherwise. It's also a number we've already cut substantially. pg_tde 2.2.2 halves it, to roughly 20% in the same worst-case test. The improvement comes from optimizing how encryption interacts with the WAL locking path, reducing the time spent doing encryption work while the lock is held.

A deeper architectural fix is already understood and in active development: encrypting each WAL record after it's constructed but before the backend acquires the lock, rather than during the locked window. That approach removes most of the serialization cost entirely, at the price of encrypting in smaller, less bandwidth-efficient chunks, a tradeoff the team is actively tuning. It isn't shipped yet. We're describing the roadmap here rather than claiming a result we haven't measured.

The takeaway for evaluation purposes: if your production PostgreSQL environment is not sustaining gigabytes per second of WAL across dozens of concurrent high-throughput sessions, this worst case does not apply to you. If it is, ask any TDE vendor, including us, for their high-concurrency WAL throughput numbers before you commit. Most won't have an answer, because most never ran the test.

Why We're Publishing the Worst Case, Not Just the Best One

A vendor can always produce a graph showing near-zero TDE overhead. It isn't hard. Standard benchmark suites simply don't generate the conditions where encryption cost shows up: they don't sustain multi-gigabyte-per-second WAL volume, and they don't run single-threaded bulk rewrites against fast enough storage to matter. Any TDE implementation looks free under those conditions, which makes a single flattering graph close to meaningless as evidence.

That's why the numbers in this whitepaper come from hardware chosen specifically to expose overhead: a 32-core AMD Threadripper 3970X paired with an Intel Optane P5800X, storage fast enough to make encryption, not disk I/O, the limiting factor. The scenarios are reproducible, the hardware is documented, and the worst cases are published next to the good ones.

This also isn't a snapshot we're leaving static. pg_tde 2.2.2 already halved the worst-case WAL overhead compared to prior releases, and the architectural fix described above is in active development. Performance on the hard cases is a continuing focus, not a one-time benchmark exercise for a launch announcement.

"When a vendor shows you a single graph proving their encryption is free, ask them about high-concurrency WAL throughput. We already published our answer."

Conclusion

Encryption at rest in PostgreSQL with pg_tde costs nothing you'll notice in the workloads most teams actually run. Relation file encryption sits inside measurement noise for standard OLTP traffic, and full end-to-end overhead, including WAL, stays under 10% in every realistic configuration tested. The cost that does exist is confined to two identifiable corners: bulk table rewrites during maintenance windows, and sustained, extreme-concurrency WAL volume far beyond typical production traffic, a cost pg_tde 2.2.2 already cut in half with further architectural improvements underway.

None of that comes from a single flattering benchmark. It comes from testing on hardware built to expose the worst case, publishing that worst case alongside the typical case, and continuing to invest in closing the gap. If you're deciding whether TDE belongs in your PostgreSQL environment, that's the standard of evidence to ask every vendor for, including Percona.

Next Steps

Encryption at rest in PostgreSQL with pg_tde costs nothing you'll notice in the workloads most teams actually run. Relation file encryption sits inside measurement noise for standard OLTP traffic, and full end-to-end overhead, including WAL, stays under 10% in every realistic configuration tested. The cost that does exist is confined to two identifiable corners: bulk table rewrites during maintenance windows, and sustained, extreme-concurrency WAL volume far beyond typical production traffic, a cost pg_tde 2.2.2 already cut in half with further architectural improvements underway.

None of that comes from a single flattering benchmark. It comes from testing on hardware built to expose the worst case, publishing that worst case alongside the typical case, and continuing to invest in closing the gap. If you're deciding whether TDE belongs in your PostgreSQL environment, that's the standard of evidence to ask every vendor for, including Percona.

If your team is evaluating encryption at rest for PostgreSQL, don't stop at a single benchmark graph, ours or anyone else's.

Talk to a Percona expert about designing an encryption strategy, including key management integration (HashiCorp Vault, Thales CipherTrust, Fortanix SDKMS, OpenBao, Akeyless) and whether your workload falls into the bulk-rewrite or high-WAL-volume corner cases covered above.

TALK TO AN EXPERT