
# REDUCING DATA ERASURE LATENCY VIA STORAGE-SIDE ENCRYPTION

PER-FILE AES-XTS KEYS + SELECTIVE PROGRESSIVE-JPEG ENCRYPTION — DELETE A FILE SECURELY IN O(1), INDEPENDENT OF ITS SIZE.

# Introduction


As data security becomes a central concern, securely deleting data has become equally critical.

Sensitive data often lives as images and scanned documents:

- Scanned bank statements & account forms
- Scanned ID cards / contracts
- Medical records & receipts (scanned)
- Personal photos with location/face data
- Cached web images in temp files

**However, truly erasing data — not just unlinking it — remains hard to do efficiently and provably on modern SSDs.**

# Is deleted data really gone?

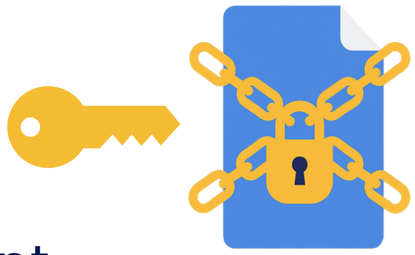

**Logical delete** → unlink only, data is still physically exists

**Physical overwrite** → O(n) slow, and unreliable on flash (wear-leveling silently remaps blocks, leaving the original behind).

**Drive-level crypto-erase** → fast, but all-or-nothing (one key wipes the whole disk).

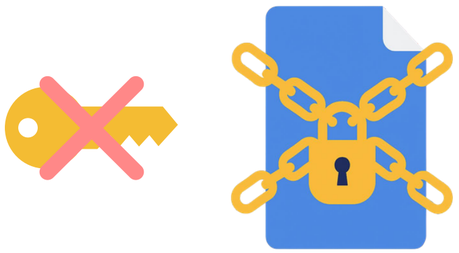
# Motivation


## This Work — Per-File Crypto-Erase

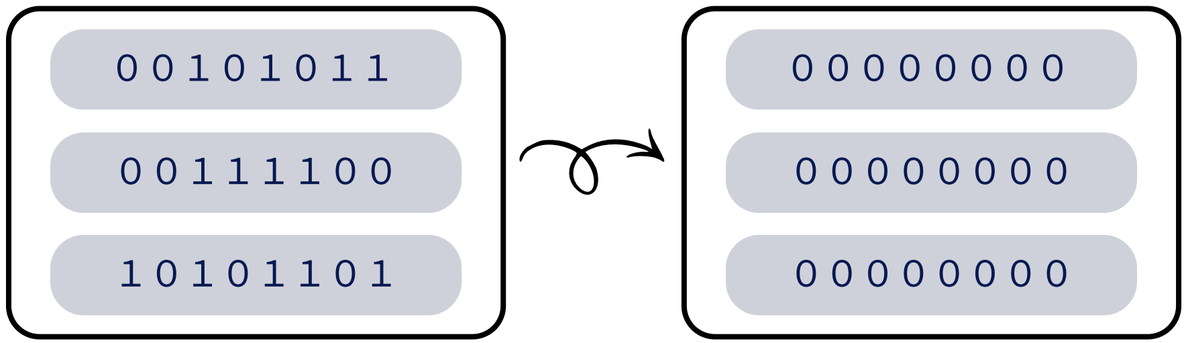


- One AES-128-XTS key per file
- Delete = discard one key → O(1), size-independent
- Verified irrecoverable (forensic + OCR attack)

**Delete key (ours)** **vs** **Software Overwrite**

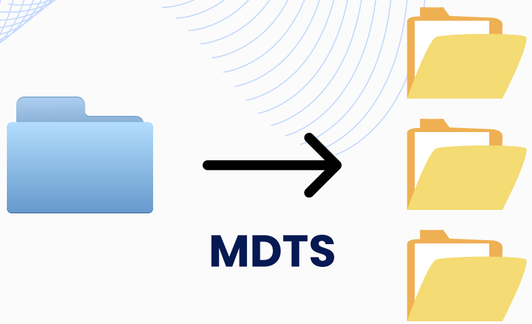


**O(1)**




**O(n)**

# NVMe Submission Queue Entry (SQE)




we put file_group_id here

| CDW0 | CDW1 | CDW2 | CDW3 | CDW4 | CDW5 | CDW6 | CDW7 | CDW8 | CDW9 | CDW10 | CDW11 | CDW12 | CDW13 | CDW14 | CDW15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Opcode + CID | NSID | file_grp (high) | file_grp (low) | MPTR | MPTR | DPTR | DPTR | DPTR | DPTR | SLBA | SLBA | NLB | cmd | cmd | cmd |

file_group_id (64-bit, inode-based)

MPTR

DPTR (PRP / SGL)

SLBA

NLB

cmd-specific

**NVMe Submission Queue Entry = 64 bytes = 16 Dwords = 512 bits**

*1 Dword (Double Word) = 4 bytes = 32 bits · CDW2 & CDW3 are Reserved in the standard NVM Write command*

# Architecture


**Host Layer**

1 **Format-Aware Classifier**
- P1 Header → Write (encrypt flag, bit 15 = 1)
- P2 Body → Write (encrypt flag, bit 15 = 1)
- P3 Rest → Write

NVMe **Commands**
(Write / Read / Sanitize / DSM)

**CDW2 | CDW3**
file_group_id

**QEMU / KVM + FEMU Emulator**

3 **NVMe Controller**
- Parse NVMe opcode (Encrypt flag)
- Generate per-file AES key

Data Path (DMA)

Control Path (key)

4 **SED Engine**
AES-128-XTS
Write → encrypt
Read → decrypt

**FTL**
- Key Mgmt — SRAM Buffer
- **Key Mapping** (LPN → Key_id → Key)
- NAND Latency Simulation

Key Flush

**Virtual TLC 3D NAND Storage**

| P1 Header | P2 Body | P3 Rest | Key Pages |
|---|---|---|---|
| Encrypted | Encrypted | Plain | NAND-reside |

**Key Mapping (two-level)**

| LPN | Key_id |
|---|---|
| 300 | 100 |
| 301 | 100 |
| 302 | 0xFFFFFFFF |

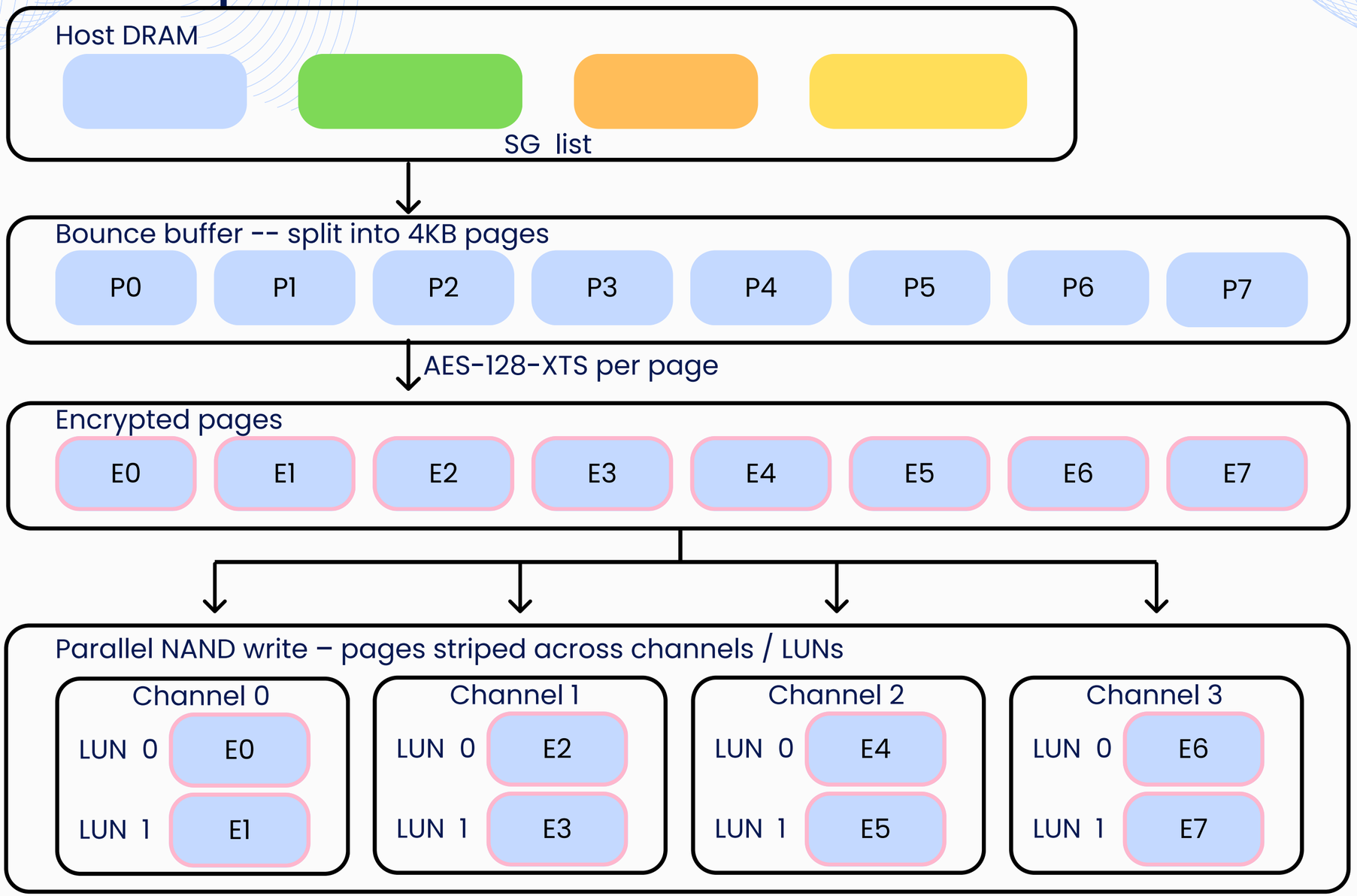
| ID | Key (32 bytes) |
|---|---|
| 100 | 0x86DFD43C… |
| 101 | 0x4165CAB8A… |

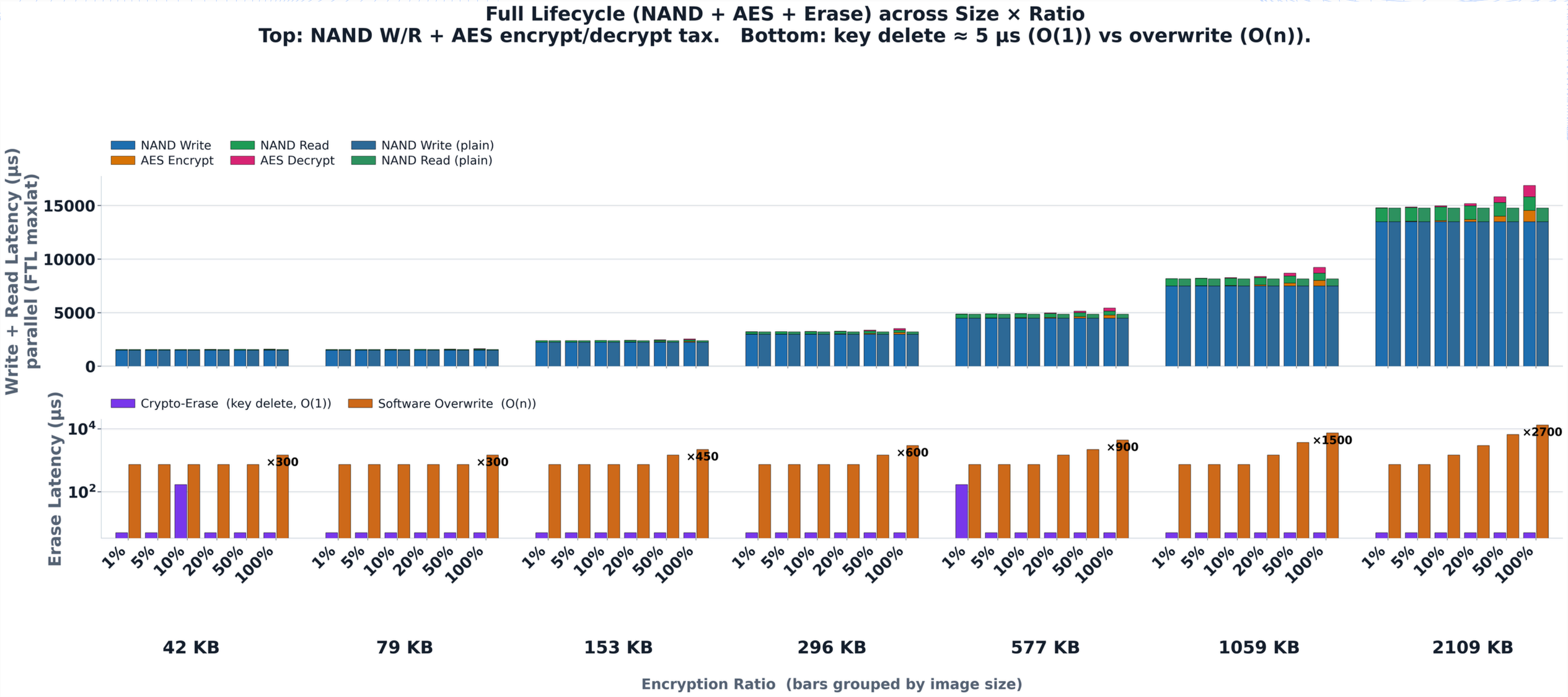
*0xFFFFFFFF = no key (plaintext / erased)*

**CRYPTO-ERASE (this work)**

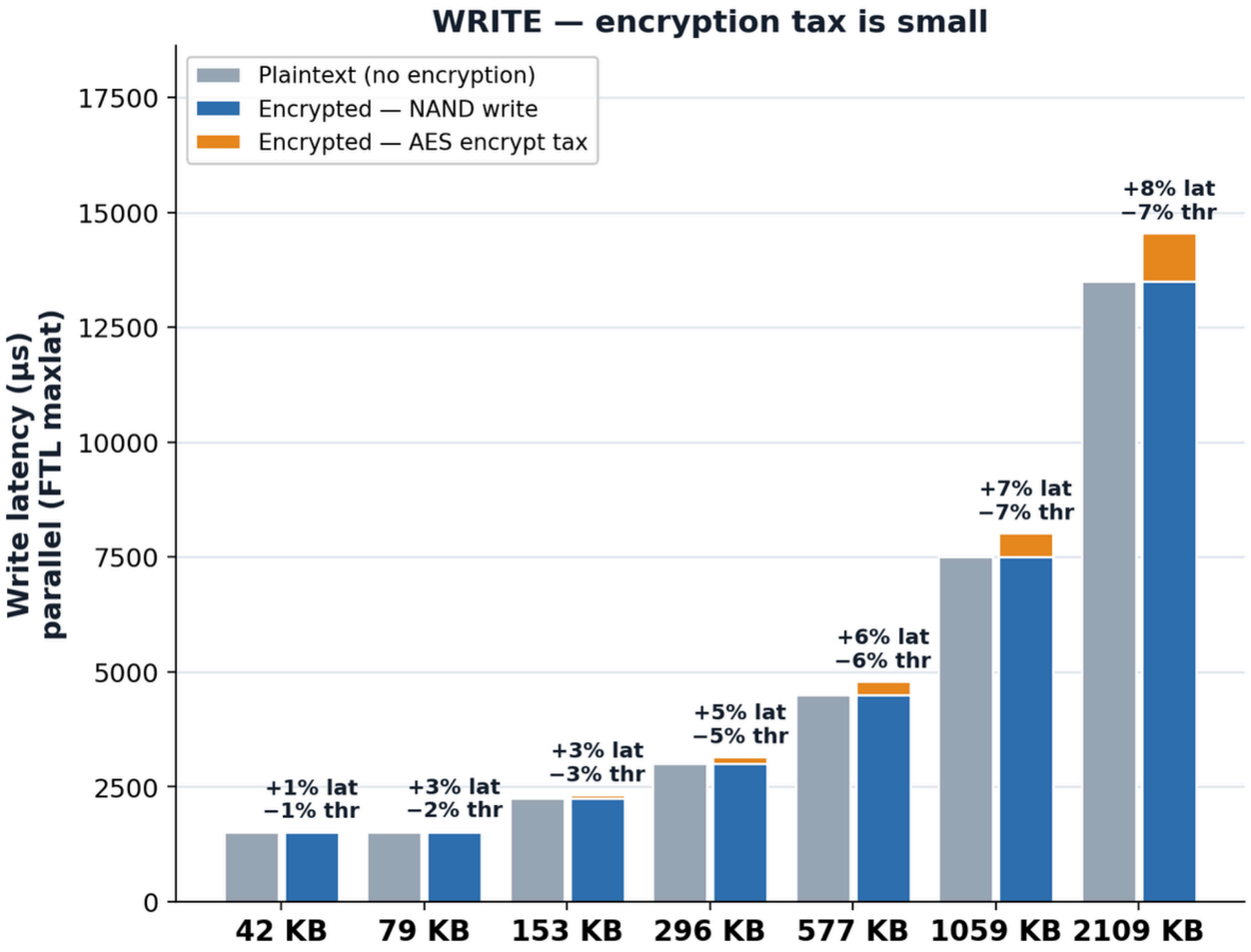
Delete file → DSM / Sanitize
→ discard one key (Key_id ← 0xFFFFFFFF)
→ O(1) · forensically unrecoverable

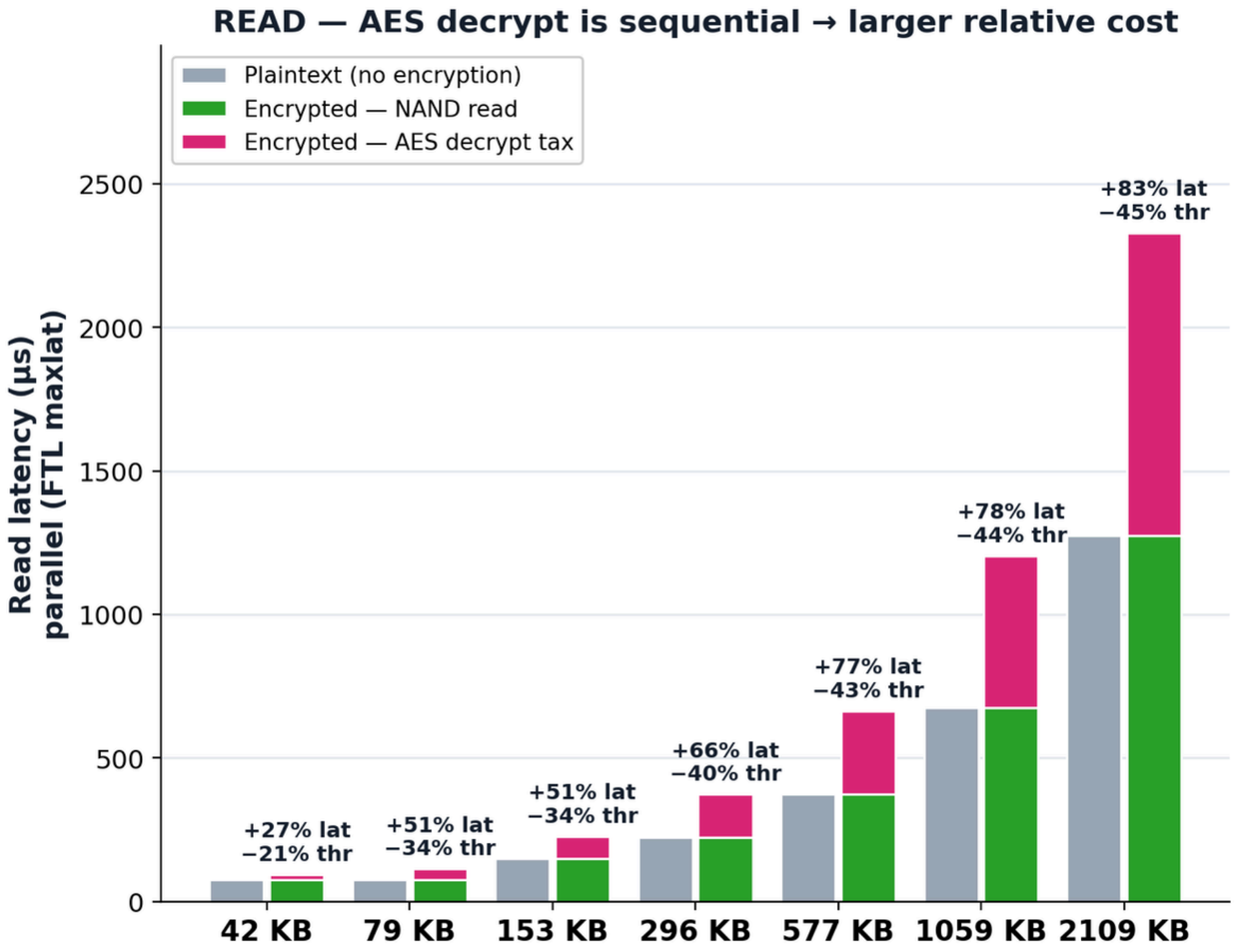
# SSD Write Pipeline


Full Lifecycle (NAND + AES + Erase) across Size × Ratio
Top: NAND W/R + AES encrypt/decrypt tax.  Bottom: key delete ≈ 5 µs (O(1)) vs overwrite (O(n)).
NAND Write
AES Encrypt
NAND Read
AES Decrypt
NAND Write (plain)
NAND Read (plain)
Write + Read Latency (µs)
parallel (FTL maxlat)
15000
10000
5000
0
Crypto-Erase  (key delete, O(1))
Software Overwrite  (O(n))
Erase Latency (µs)
10^4
10^2
×300
×300
×450
×600
×900
×1500
×2700
1%
5%
10%
20%
50%
100%
42 KB
79 KB
153 KB
296 KB
577 KB
1059 KB
2109 KB
Encryption Ratio  (bars grouped by image size)

# Encrypted vs Plaintext Read/Write Performance — at 100% encryption




*lat = latency increase vs plaintext · thr = throughput drop. NAND work is identical; the only added cost is the AES tax.*

# New questions arise


- How little of a file must we encrypt to guarantee it cannot be recovered — while keeping read/write latency low?
- Encrypt too much → **slower I/O**
- Encrypt too little → **an attacker recovers it.**

In this work, we use **Progressive JPEG** as our study format to evaluate both the latency overhead and the security of per-file selective encryption.


# Evaluation — Security (Worst-Case Attacker)


## Threat model — worst-case forensic adversary

- Given **header + surviving data**, reconstructs the destroyed DC (brightness) layer from surviving AC (texture)
- Inter-block continuity → a Poisson solve ; more iterations = stronger attacker (rigorous: 1500)

## Verifying the secure-erase percentage

- Sweep encryption ratio 1 % → 100 %
- Secure encryption ratio = smallest ratio at which ≥ 90 % of worst-case samples fall below threshold

### Photographs

**Pearson correlation** between the forensic-recovered image and the original (0–1; lower = more destroyed). Secure if **< 0.30**.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

### Documents

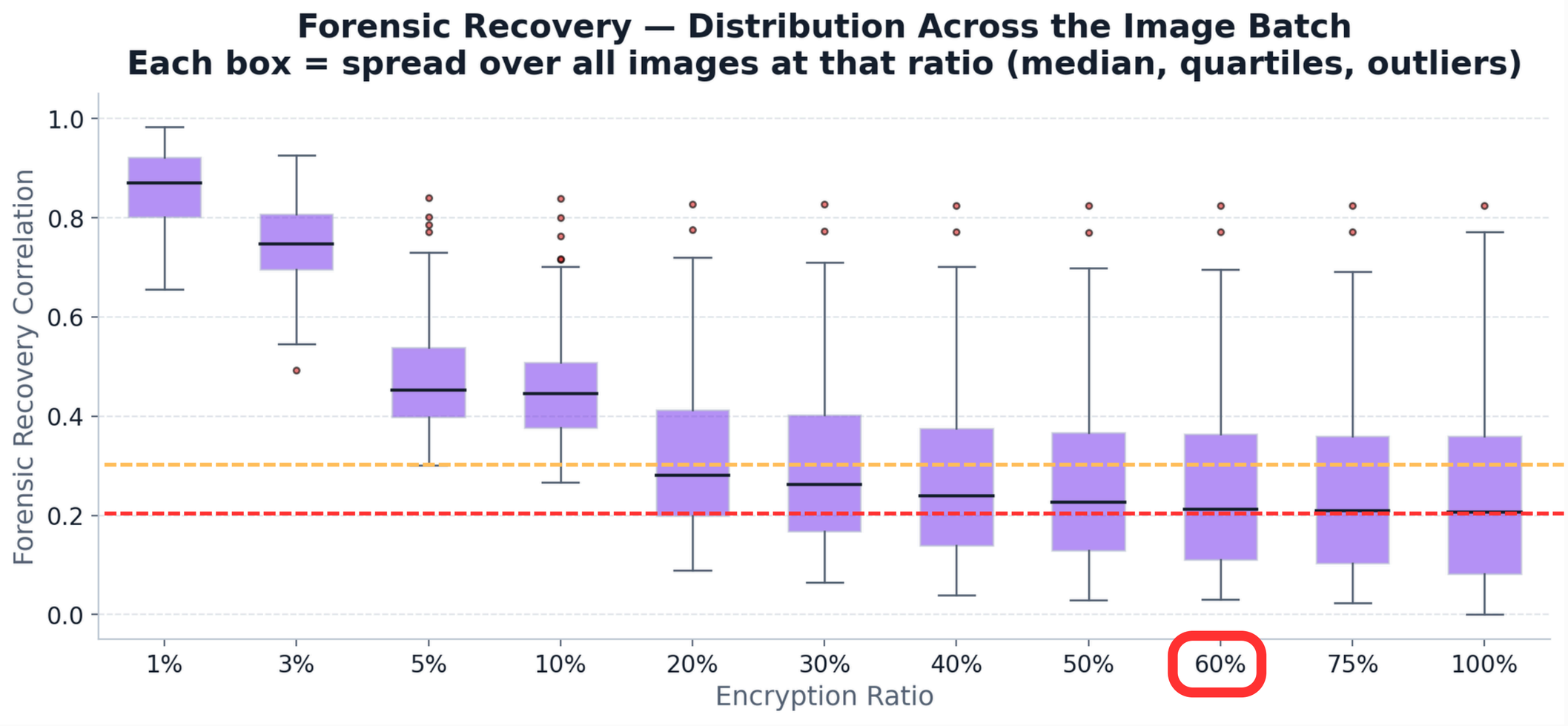
Dataset: FUNSD

**OCR character-recovery** = 1 - normalized Levenshtein(ground-truth, OCR). Secure if **< 0.10**.

$$\text{recovery} = 1 - \frac{\text{Lev}(GT,\, OCR)}{\max(|GT|,\, |OCR|)}$$

**Amount·Date·Account·Name**

# Results – Photos

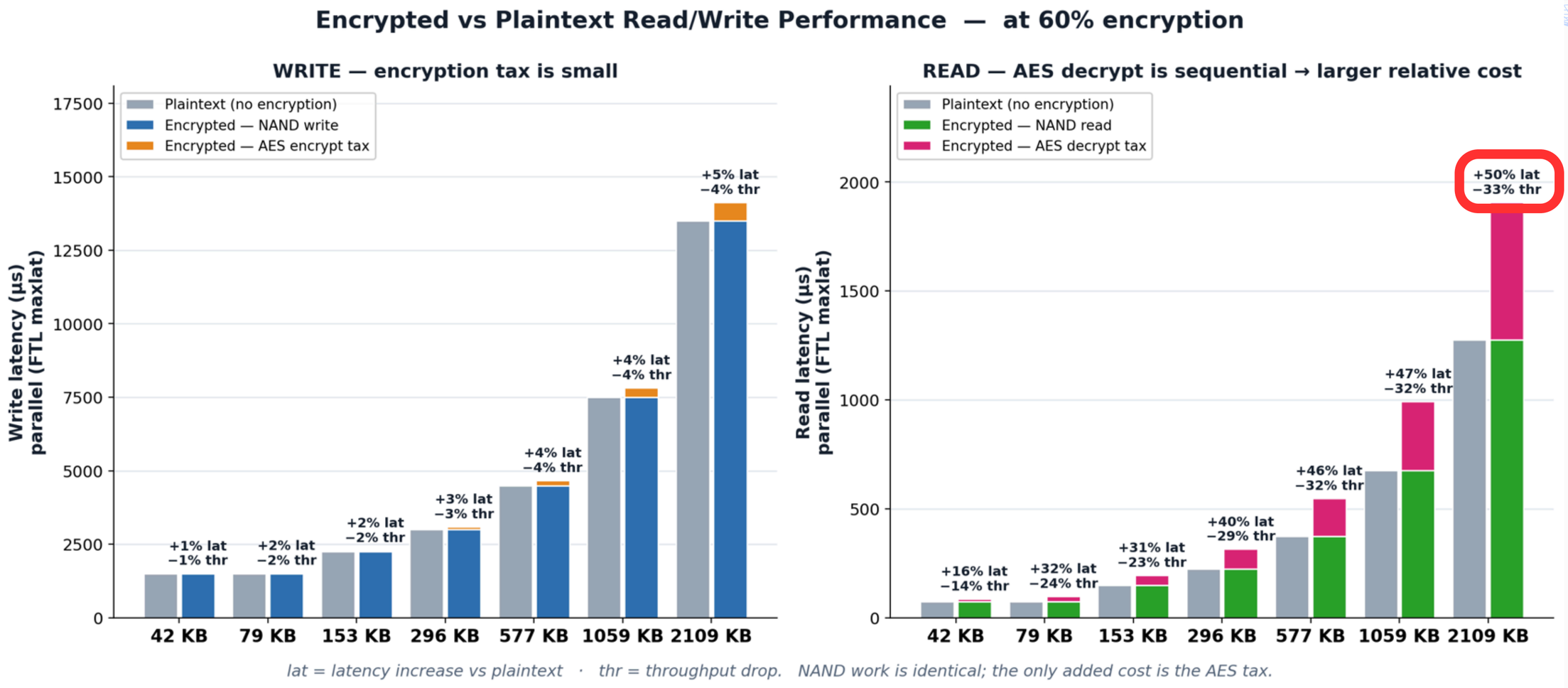


# Results – Photos

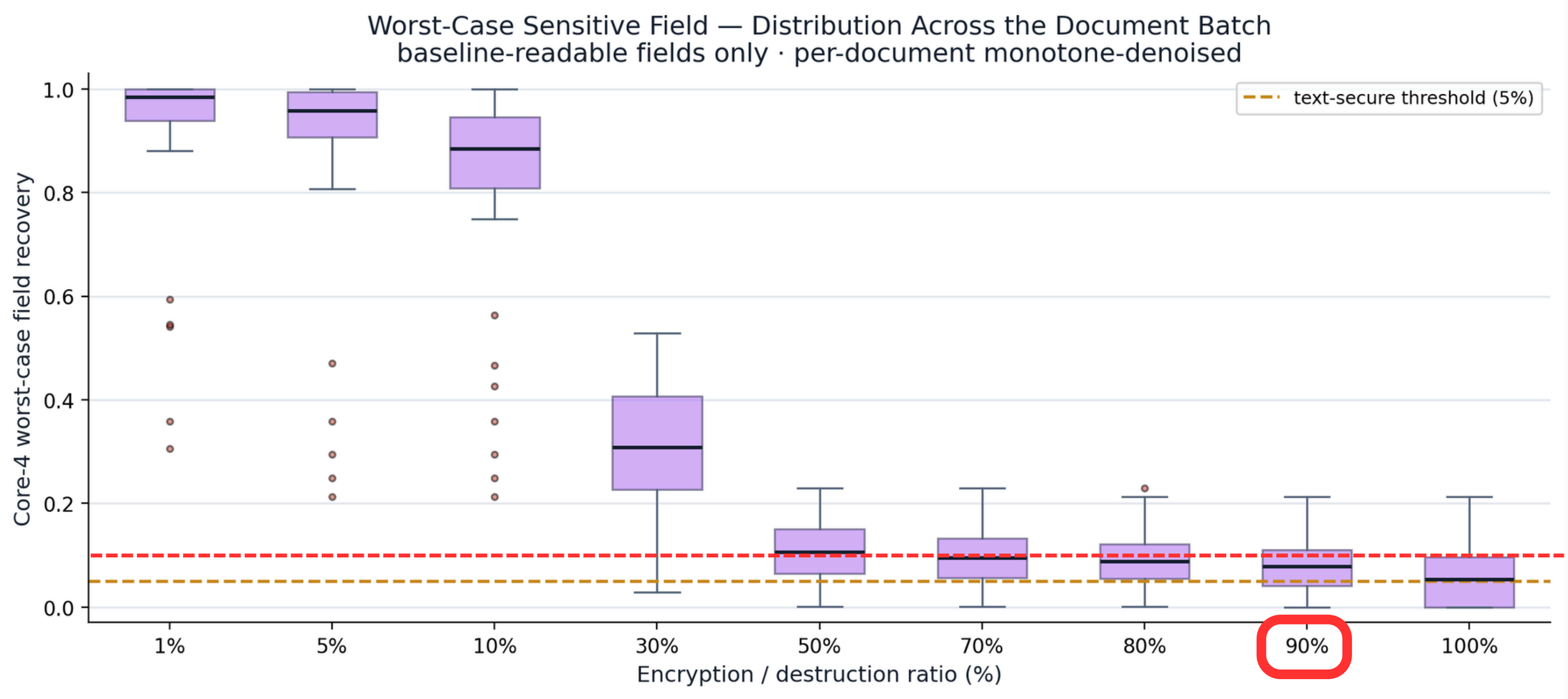
1% 3% 5% 10% 20% **40%** 60%

1% 3% 5% 10% 20% 40% **60%** 75%

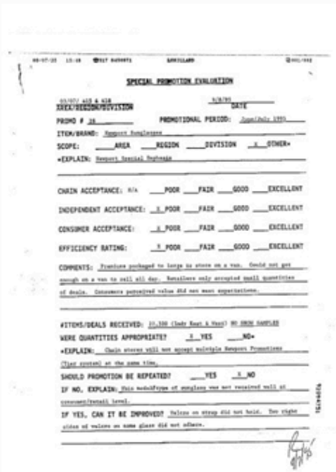
# Results – Photos



# Results – Documents

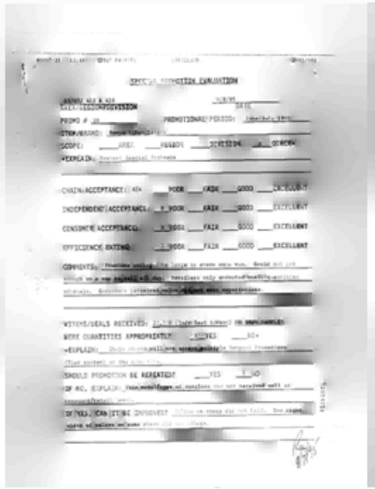


# Results – Documents

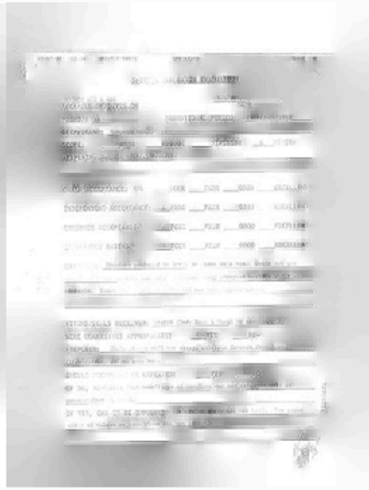


0%

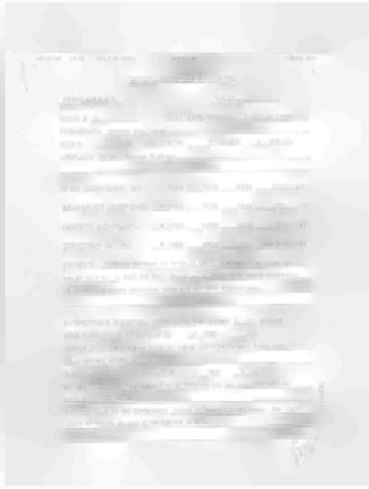
1%



5%



10%



30%

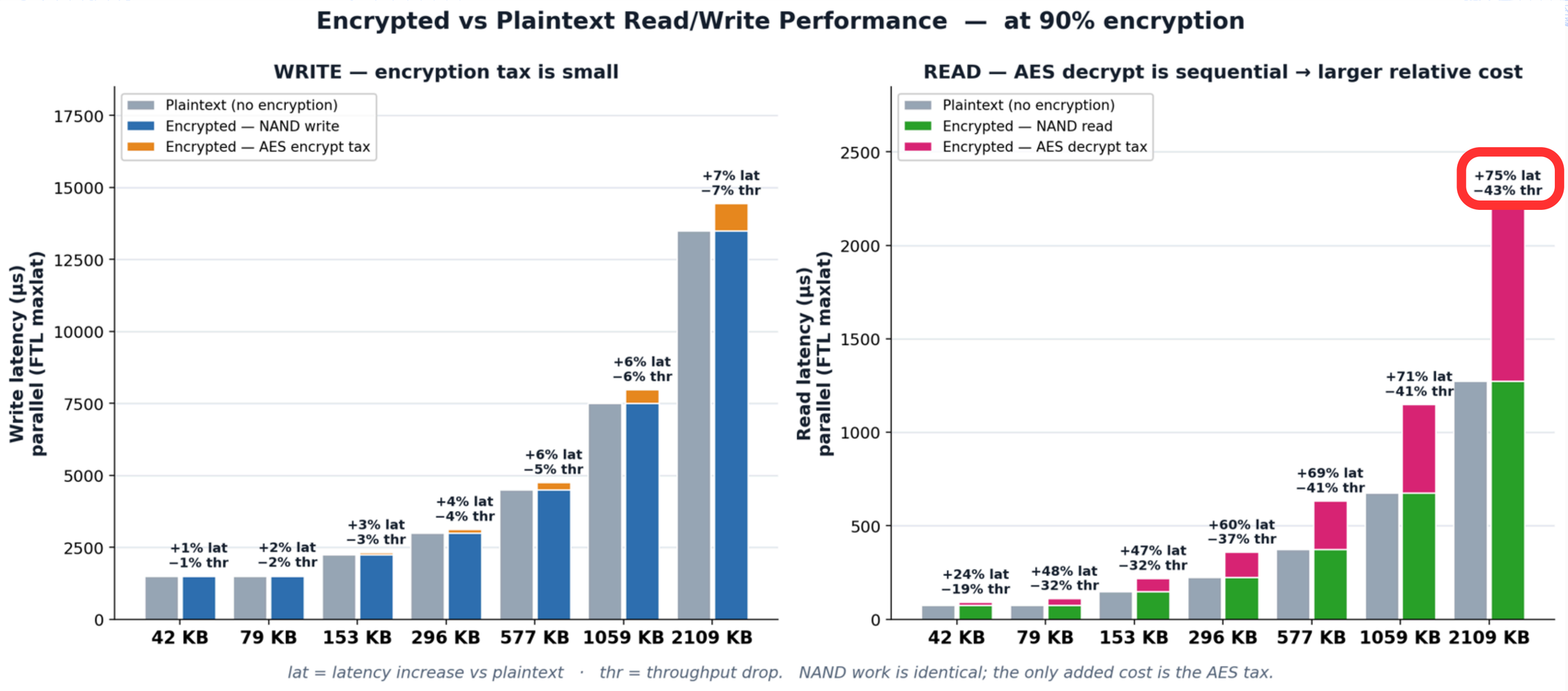


60%



**90%**

# Results – Documents



# Conclusion


- O(1) per-file deletion
- Verified, not assumed
- Quantified: photos ≈60% · docs ≈90%
- Read-side AES-decrypt cost surfaced

**Future Work**

- Parallel / hardware AES (read cost)
- Reliable key destruction
- Generalize to video / PDF / other codecs

# THANK YOU