Cryptographic Kernel – Investigations on possible leaks in Kyber (ML-KEM in FIPS 203) reference implementation

kyber\_bin was generated as a Linux target using gcc -O3 -march=native -flto -ftree-vectorize -o kyber\_bin kyber\_bin.c kyber\_api.c kem.c indc  
pa.c fips202.c cbd.c ntt.c symmetric-shake.c verify.c poly.c polyvec.c randombytes.c reduce.c

Valgrind Memory Check – A thorough analysis of memcheck on the kyber\_bin file in our repository indicated no memory leakage. All heap blocks allocated were freed as expected.

Valgrind Cache check – Output of the cachegrind tool on the binary indicated some irregularities worth investigating. Output can be seen in human readable format using

cg\_annotate cachegrind.out.5049

Perf –

Command - perf stat -e cache-misses,branch-misses,cycles ./kyber\_bin

Output - Performance counter stats for './kyber\_bin':

20,353 cache-misses

11,134 branch-misses

28,23,867 cycles

0.003148668 seconds time elapsed

0.003134000 seconds user

0.000000000 seconds sys

Branch misses in this output are especially worth investigating. To investigate further, consider running perf stat -e cache-references,cache-misses on isolated components of Kyber’s code (e.g., NTT or polynomial multiplication functions) to see if the cache misses are localized to specific cryptographic primitives. You could also use Cachegrind to help pinpoint exactly where the highest misses are occurring in your code.

Tools like Callgrind and further inspection in perf record -e branches,branch-misses may help you identify which parts of Kyber’s code have the most branch mispredictions. A high count in specific functions could suggest secret-dependent branching.