PERFORMANCE REPORT FOR PROJECT 2

MD5 vs SHA256 for Rainbow Table Generation:

- **MD5:** Exhibited an average processing time of approximately 102.80 ms to 103.46 ms across tests. When increasing the thread count from 4 to 10, the time slightly decreased, showing mild performance improvements due to increased parallelism.
- **SHA256:** Showed a slightly longer average processing time, ranging from 106.06 ms to 106.98 ms. However, like MD5, it exhibited reduced processing times with increased threading, suggesting that SHA256 also benefits from parallel processing, but the times are consistently higher than MD5, indicating that SHA256 is computationally more demanding.

MD5 vs SHA256 for Hash Cracking:

- **SHA256:** The cracking time ranged from 1.6631 ms to 1.6687 ms. There was a notable performance regression with increased threading, suggesting that SHA256 may not scale linearly with the increase in threads for this particular task.
- **MD5:** Demonstrated quicker cracking times ranging from 1.5310 ms to 1.5332 ms. Similar to SHA256, the threading impact was minimal, and MD5 remained consistently faster than SHA256 in hash cracking operations, which is expected given the less complex nature of the MD5 algorithm.

Impact of Threading

Rainbow Table Generation:

- Threading seems to benefit both MD5 and SHA256 in generating rainbow tables. The performance improvement from 4 threads to 10 threads is marginal but noticeable, which suggests that the operation scales somewhat linearly with the number of threads.
- However, the performance regression in some SHA256 tests with higher threads hints that there might be diminishing returns or inefficiencies introduced at higher levels of parallelism, possibly due to overhead from thread management or synchronization.

Crack Hashes Operation:

- For SHA256, increased threading initially shows performance improvement, which is a good indication of the algorithm's capability to utilize parallel computing resources effectively. However, the results with 10 threads do not show a proportionate decrease in time, suggesting that the application might be hitting a bottleneck elsewhere, such as I/O operations or memory access speeds.
- MD5 also shows minimal change with increased threading, maintaining similar times across different thread counts, indicating a potential upper limit on the benefits obtained from threading for this specific operation.