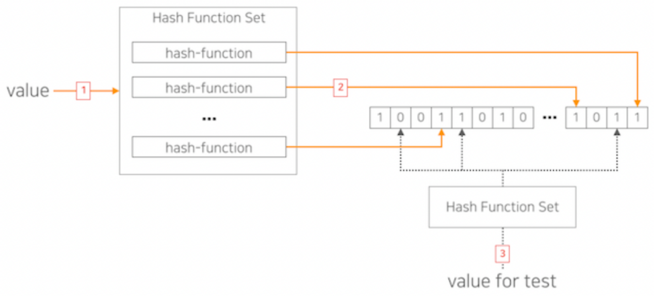
CSED415-01 Project 2: Proactive Password Checker

20170252 Ko Heeju
posko17@postech.ac.kr

1. Background

A Bloom filter is a probabilistic data structure used to test whether an element is a member of set. To add an element to the Bloom filter, a designated number of hash functions are applied to the element. The hashed values are used to set the corresponding bits in the bit array to 1.



source: https://meetup.nhncloud.com/posts/192 [Accessed Mar 26, 23]

As we can see, hashed value will set to bit 1 of corresponding index. The hashed value is an index of Bloom filter bit array.

A Bloom filter structure allows fast set membership tests. The test can be performed in constant time of O(k) where k is the number of hash functions used to Bloom filter. However, it has limitations. It has a possibility of false positive. Even you assume that the hash functions in Bloom filter provide uniform distribution, if the test cases are larger than the size of Bloom filter bit array, then false positive must exists. The proof is provided with pigeonhole principle.

2. Design

For the sake of efficiency and consistency, hash functions should be chosen carefully. XXH3, XXH128, Murmur3 are non-cryptographic functions that provide good performance and collision resistance. However, both hash functions are not the best choice to handle large inputs.

XXH function provide better collision resistance and bandwidth than Murmur3 [1][2]. Although, the performance will vary between CPU architectures, we can say that xxh3 generally shows better performance according to the provided benchmarks.

For the sake of security, using non-cryptographic hash functions may be risky to brute force attack. Therefore, combining cryptographic function will provide stronger security. SHA-256, SHA-3, blake2, whirlpool are the cryptographic functions. On x86-based processors, blake2 showed highest performance. On ARM-based processor, the result will

[1] https://github.com/Cyan4973/xxHash [Accessed Mar 26, 23]
[2] https://github.com/rurban/smhasher [Accessed Mar 26, 23]

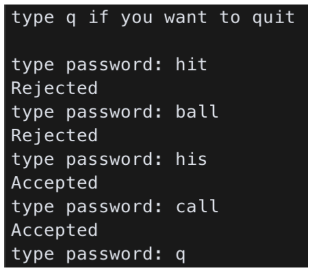
depend on the other factor. On Macbook Pro M1, SHA-256 was 11% faster and showed less collision. Therefore, we chose SHA-256.

To improve performance, we combine XXH3 and SHA-256. By applying Kirsch-Mitzenmacher-Optimization, we can compute only 2 hash functions. We chose third hash function by adding the result of XXH3 and the result of SHA-256.

Therefore, we chose 3 hash functions: XXH3, SHA-256, XXH3 + (7*SHA-256).

3. Result

There are two ways to use this program. First, you can execute the program with "python3 Bloomfilter.py dictionary.txt" or "./Bloomfilter dictionary.txt". Then you can check whether the password is accepted or rejected.



```
type q if you want to quit

type password: hit
Rejected
type password: ball
Rejected
type password: his
Accepted
type password: call
Accepted
type password: q
```

The Bloom filter is built with provided dictionary.txt. It will show the result whether the input is accepted or rejected. Also, the result is stored in Bloomchecked.txt.

Second, you can execute the program with "python3 Bloomfilter.py dictionary.txt test_candidate.txt" or "./Bloomfilter dictionary.txt test_candidate.txt". It automatically checks the words in test_candidate.txt and the result will be saved in Bloomchecked.txt.

We operated test with all possible word of length 3~6. You can generate all words using TestCandidateGen.py. The program will generate test_candidate.txt file. Our test used the file generated by TestCandidateGen.py.

If we assume the distribution is perfectly uniform, the false negative rate will be nearly 0.0625[3]. The result of our design shows 0.0631. By trying all possible word of length 3~6, it showed 20273223 false positives out of 321271704 tries. Our design had 1% higher collisions than optimized result.

4. Conclusion

We can say that our result is good enough to use as a Bloom filter. The collision rate was only 1% higher than optimized result. Also, we can say that our implementation is fast enough in that XXH3 showed best performance according to the benchmarks on several architectures including ARM based CPU. SHA-256 showed better performance than blake2 on ARM architecture even blake2 showed best performance on x86 architectures. Furthermore, we applied Kirsch-Mitzenmacher-Optimization to calculate only 2 hash functions rather than 3.

However, we can still improve our implementation when considering speed. We can choose non-cryptographic hash functions instead of SHA-256 or blake2. Since our parameter N is 8192 which is small, applying cryptographic hash functions might show less performance when it comes to collisions. As we have described above, even the optimized result had 6.25% collisions. Therefore, we could have used non-cryptographic hash functions to improve speed.

[3] https://hur.st/bloomfilter/?n=1380&p=&m=8192&k=3 [Accessed Mar 26, 23]