Software and Cybersecurity Lab

CS445 Lab1

Name: Dipean Dasgupta ID: 202151188

Task: Write a program (in C/C++/Java/Python) to compare the execution time of the following hash functions.

MD5; SHA-1; SHA-256; SHA-512; MurmurHash3; City Hash; Farm Hash; Whirlpool; BLAKE2s.

Programming Language: Python version: **3.10.12**

Setup:

- **Platform:** Google Colab
- **Libraries/Packages:** mmh3, cityhash, pycryptodome, whirlpool.
- **Input size:** 1000 characters (8000 bits)
- **Input Strings:** 10
- **Output:** Digest as per hash function ranging from 64 to 512.

Code Overview and Explanation:

The code to compare the execution of hash function in Python starts with importing the inbuilt hash function from the packages installed.

In next step a random string of 1000 characters were generated which included ascii letters, digits, punctuations. The string is then converted to bytes, which is needed for the hash functions that operate on byte data. Total of 10 random strings were generated.

All the hash functions in the task were called from the inbuilt libraries for execution. The data which is the input string/ message was passed as argument to all those hash functions.

A list **execution_times[]** is created to store the execution times of each hash function.

For each hash function (name, function, and output size):

- **times = []**: list to store the execution times for the current hash function.
- **Inner loop**: For each hash function loop goes for 10 random strings generated:
  - **start_time = time.time()**: Recording the start time.
  - **func(rand_data)**: Calling the hash function with the data.
  - **end_time = time.time()**: Recording the end time.
  - **times.append(end_time - start_time)**: Calculating and storing the time taken for this execution.
- **avg_time = statistics.mean(times)**: Computing the average execution time over the 10 runs.
- **execution_times.append((name, output_size, avg_time))**: Storing the hash function name, output size, and average time in the **execution_times[]** list.

The **execution_times** list is then sorted by average execution time in ascending order. The sorted result is then printed in a tabular format, showing the serial number, hash function name, language, input length, digest length, average time, and priority.

Result Table:

Priority	Hash Algorithm	Language	Input Size(bits)	Output Size(bits)	Avg. Time(s)
1	CityHash	Python	8000	64	0.0000015020
2	FarmHash	Python	8000	64	0.0000015030
3	MurmurHash3	Python	8000	128	0.0000018358
4	SHA-1	Python	8000	160	0.0000050068
5	SHA- 512	Python	8000	512	0.0000061512
6	SHA-256	Python	8000	256	0.0000078440
7	MD-5	Python	8000	128	0.0000082254
8	Whirlpool	Python	8000	512	0.0000226259
9	BLAKE2s	Python	8000	256	0.0004254580

Code of Execution:

```
pip install mmh3 cityhash pycryptodome
pip install whirlpool
import hashlib
import mmh3
import cityhash
from farmhash import FarmHash32, FarmHash64, FarmHash128
from Crypto.Hash import BLAKE2s, SHA512, SHA256, MD5, SHA1
import whirlpool
import time
import statistics
import random
import string

# Defining All hash functions one by one
def hash_md5(data):
    return hashlib.md5(data).digest()

def hash_sha1(data):
    return hashlib.sha1(data).digest()

def hash_sha256(data):
    return hashlib.sha256(data).digest()

def hash_sha512(data):
    return hashlib.sha512(data).digest()

def hash_murmur3(data):
```

```
    return mmh3.hash_bytes(data)

def hash_cityhash(data):
    return cityhash.CityHash64(data)

def hash_blake2s(data):
    return BLAKE2s.new(data=data).digest()

def hash_farmhash64(data):
    return FarmHash64(data)

def hash_whirlpool(data):
    return whirlpool.new(data).digest()

# Listing of hash functions for comparison
hash_functions = [
    ("MD5", hash_md5, 128),
    ("SHA-1", hash_sha1, 160),
    ("SHA-256", hash_sha256, 256),
    ("SHA-512", hash_sha512, 512),
    ("MurmurHash3", hash_murmur3, 128),
    ("CityHash", hash_cityhash, 64),
    ("FarmHash", hash_farmhash64, 64),
    ("Whirlpool", hash_whirlpool, 512),
    ("BLAKE2s", hash_blake2s, 256)
]

# Generating a random string of 1000 characters
def generate_random_text(length=1000):
    return ''.join(random.choices(string.ascii_letters + string.digits +
string.punctuation + " ", k=length))
# Preparing a large string (at least 1000 characters)
data = [generate_random_text(1000).encode() for _ in range(10)]
# Recording execution times
execution_times = []

for name, func, output_size in hash_functions:
    times = []
    for rand_data in data:
        start_time = time.time()
        func(rand_data)
        end_time = time.time()
        times.append(end_time - start_time)
    avg_time = statistics.mean(times)
    execution_times.append((name, output_size, avg_time))

# Sorting by execution time to determine priority
execution_times.sort(key=lambda x: x[2])
```

```
# Display of results in tabular form
input_size_bits = len(data[0]) * 8
print(f"{'S.No.':<5} {'Hash Function':<20} {'Language':<10} {'Input Size
(bits)':<20} {'Output Size (bits)':<20} {'Time (s)':<15} {'Priority':<8}")
for idx, (name, output_size, avg_time) in enumerate(execution_times, start=1):
    print(f"{idx:<5} {name:<20} {'Python':<10} {len(data) * 8:<20}
{output_size:<20} {avg_time:<15.10f} {idx:<8}")
```

Comparison Output:

S.No.	Hash Function	Language	Input Size (bits)	Output Size (bits)	Time (s)	Priority
1	CityHash	Python	8000	64	0.0000015020	1
2	FarmHash	Python	8000	64	0.0000015020	2
3	MurmurHash3	Python	8000	128	0.0000018358	3
4	SHA-1	Python	8000	160	0.0000050068	4
5	SHA-512	Python	8000	512	0.0000061512	5
6	SHA-256	Python	8000	256	0.0000078440	6
7	MD5	Python	8000	128	0.0000082254	7
8	Whirlpool	Python	8000	512	0.0000226259	8
9	BLAKE2s	Python	8000	256	0.0004254580	9

Comparison Analysis:

As we know in general, non-cryptographic hash functions like MurmurHash3, CityHash, and FarmHash focus on speed and efficiency, making them suitable for use cases like hash tables and checksums where security is not the primary concern. Code output depicts the same fact; **CityHash** has been the **most efficient and fastest in terms of execution time** taking only .0000008106 s. On the other hand, cryptographic hash functions like SHA-256, SHA-512, Whirlpool, and BLAKE2s are designed for security applications, providing strong resistance to collisions and other attacks. As per the output, **BLAKE2s** hash algorithm took the **least position** in terms of execution time taking .0001423120s to complete.

So, In case of Speed and Efficiency: **CityHash: Priority #1** **BLAKE2s: Priority #9**