hash-compare-test

February 19, 2020

1 Performance Comparison of SHA-256, SHA-1, Blake2b, and Blake2s

1.0.1 Author: Sae Hyong Park

[29]:
```
from hashcompare import Hash
from hashlib import sha1, sha256, blake2b, blake2s
from zlib import crc32
import time
import inspect
import numpy as np
```

1.1 Simulation Set-up

simulated_packet denotes 10 byte

x_range denotes 1 to 1000 times the size of the simulated_packet

The smallest being 10 bytes and the largest being 10000 bytes or 10k bytes

[39]:
```
x_range = [1,10,100,200,300,400,500,600,700,800, 900, 1000]

hash_mapper = [Hash(sha256(),"sha256"), Hash(sha1(),"sha1"),␣
 ↪Hash(blake2b(),"blake2b"), Hash(blake2s(),"blake2s")]
hash_mapper = np.array(hash_mapper)

simulated_packet = "abcdefghij"
packet= []
for weight in x_range:
    packet.append(simulated_packet * weight)

for idx, hash_f in enumerate(hash_mapper):

    for i in range(len(x_range)):
        hash_mapper[idx].set_start(time.time())
```

```
        for j in range(Hash.NUM_OF_ROUNDS):
            #packet = simulated_packet*i
            hash_f.update(packet[i].encode())
            hash_v = hash_f.digest()

        hash_mapper[idx].set_finish(time.time())

    print("{} {} performed {} operations in {} {} \n".format
        (idx, hash_mapper[idx].get_name(), Hash.NUM_OF_ROUNDS, ␣
↪len(hash_mapper[idx].get_duration()), hash_mapper[idx].get_duration()))
```

```
0 sha256 performed 10000 operations in 12 [0.017841815948486328,
0.02378082275390625, 0.05012679100036621, 0.07955408096313477,
0.11300134658813477, 0.1415090560913086, 0.17212891578674316, 0.203474760055542,
0.2313551902770996, 0.26607489585876465, 0.29224276542663574,
0.32123899459838867]

1 sha1 performed 10000 operations in 12 [0.01632523536682129,
0.017429828643798828, 0.03087902069091797, 0.04433608055114746,
0.06261229515075684, 0.07734298706054688, 0.09079504013061523,
0.10367703437805176, 0.11681318283081055, 0.13128972053527832,
0.14490008354187012, 0.15737104415893555]

2 blake2b performed 10000 operations in 12 [0.012406110763549805,
0.013839960098266602, 0.02892279624938965, 0.04445528984069824,
0.06334614753723145, 0.08010601997375488, 0.09444093704223633,
0.10946512222290039, 0.1251070499420166, 0.14109301567077637,
0.15746092796325684, 0.1701669692993164]

3 blake2s performed 10000 operations in 12 [0.0110321044921875,
0.012994050979614258, 0.03080916404724121, 0.04953122138977051,
0.07018923759460449, 0.09145331382751465, 0.10683798789978027,
0.12557125091552734, 0.1447770595550537, 0.1640300750732422, 0.1811048984527588,
0.201279878616333]
```

2 How many second does it take to perform one hashing?

It takes about one millionth of a second for one hashing to be performed.

[120]:
```
test_mapper = [Hash(sha256(),"sha256"), Hash(sha1(),"sha1"),␣
↪Hash(blake2b(),"blake2b"), Hash(blake2s(),"blake2s")]
test_mapper = np.array(test_mapper)

for idx, hash_f in enumerate(test_mapper):
    test_mapper[idx].set_start(time.time())
```

```
    for j in range(1000000):
        hash_f.update(packet[0].encode())
        hash_v = hash_f.digest()
    test_mapper[idx].set_finish(time.time())

for idx, hash_f in enumerate(test_mapper):
    print("performed 1 operation in {} \n".format(test_mapper[idx].
 ↪get_duration()[0]))
    #print("{0:s} performed 1 operation in {1:.2f} \n".format(test_mapper[idx].
 ↪get_name(), test_mapper[idx].get_duration()))
```

```
performed 1 operation in 1.7700560092926025

performed 1 operation in 1.5589969158172607

performed 1 operation in 1.198828935623169

performed 1 operation in 1.0709331035614014
```

2.1 Measure arithmetic calculations one million times

[125]:
```
start = time.time()
for i in range(1000000):
    finish = time.time()
    random1 = finish * 12 / 3.0 + 1234
    random2 = start * 124 / 4.2  + 234
    random3 = finish * 324 / 3.4 + 2344
    random4 = start * 2346 / 5.2 + 424324
    random6 = finish * 42310 / 13.0 + 23423432
    random7 = start * 12349 / 14.2 + 232.033
    random8 = finish * 4448 / 13.4
    random9 = start * 12347 / 25.2
    duration = finish - start

print("performed operation in {} \n".format(duration))
```

```
performed operation in 1.1086070537567139
```

[40]:
```
default_duration = np.array(hash_mapper[0].get_duration())

import pandas as pd
import matplotlib.pyplot as plt
default_duration = pd.DataFrame(default_duration)
```

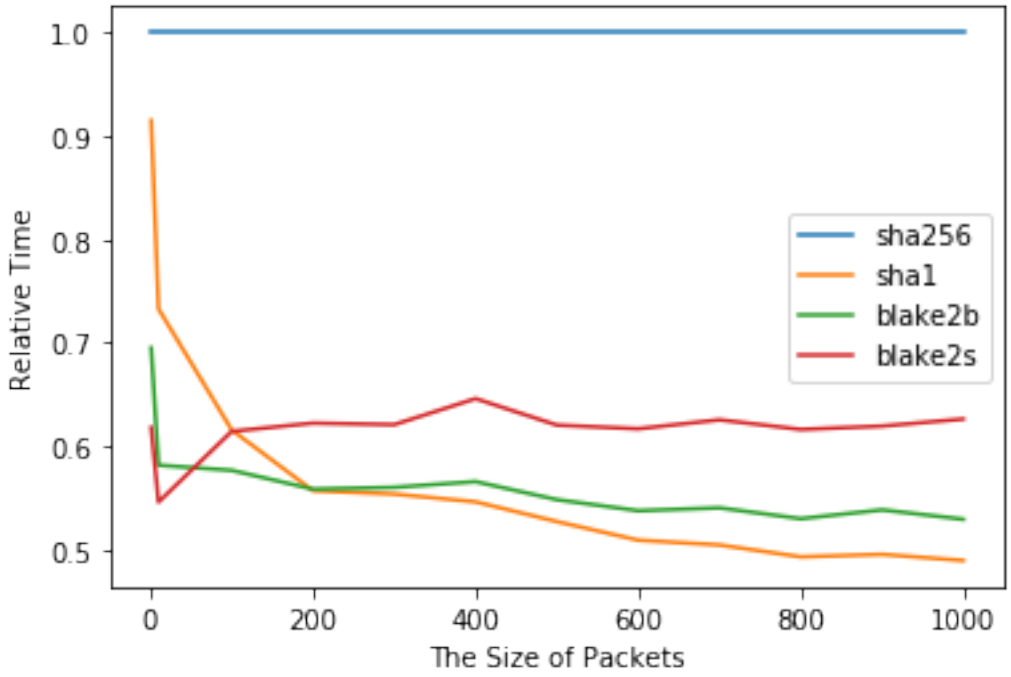
```
plot_result = {}
relative_performance = {}
for idx, hash_f in enumerate(hash_mapper):
    hash_mapper[idx].duration = pd.DataFrame(hash_mapper[idx].get_duration())
    hash_mapper[idx].duration.name = hash_mapper[idx].get_name()

    relative_performance.update({idx: hash_mapper[idx].get_duration()/
 ↪default_duration})
    #print(relative_performance[idx])
    alist = []
    for item in relative_performance[idx].values:
        alist.append(item)
        #nlist = np.asanyarray(alist)
        plot_result.update({hash_mapper[idx].get_name():alist})

plot_result =  pd.DataFrame(plot_result, index=x_range)
plot_result.sha256 = plot_result.sha256.astype(float)
plot_result.sha1 = plot_result.sha1.astype(float)
plot_result.blake2b = plot_result.blake2b.astype(float)
plot_result.blake2s = plot_result.blake2s.astype(float)
```

2.2 Graph Analysis

Sha-256 is used as the baseline performance measurement unit and the Y-axis shows the relative time they needed to perform a hashing. X-axis show the packet size and the unit is 10 bytes, meaning 200 denotes 2k bytes, 400 4k bytes and so on.

The graph shows that sha-1 has the best performance for larger packets. However, sha-1 is known to be vulnerable due to high probability of hash colision.

[42]:
```
ax = plot_result.plot()
ax.set_xlabel("The Size of Packets")
ax.set_ylabel("Relative Time")
ax.legend(loc='center right')
plt.show()
```


1.0
0.9
0.8
0.7
0.6
0.5
Relative Time
0
200
400
600
800
1000
The Size of Packets
sha256
sha1
blake2b
blake2s