International Journal of Computer Applications (0975 – 8887)
Volume 128 – No.16, October 2015

Cryptanalysis of bcrypt and SHA-512 using Distributed Processing over the Cloud

Atishay Aggarwal
VESIT,University Of Mumbai
Mumbai,India

Pranav Chaphekar
VESIT,University Of Mumbai
Mumbai,India

Rohit Mandrekar
VESIT,University Of Mumbai
Mumbai,India

ABSTRACT

Passwords are one of the commonly used method to protect one's personal information against the intruders. But storing passwords as plaintext is not safe, hence they are saved in form of hashes. And authentication occurs by comparing the hash in the database to the hash generated from input taken. It is crucial that the hashing algorithm is not only tough to reverse engineer but, should also be nearly impossible to find a collision [1]. The study considers a different approach using distributed processing to compute multiple hashes at a very high speed, making one of the most widely used hashing algorithm SHA- 512[2] seem not that secure after all. The approach involves cryptanalyzing bcrypt, another hashing algorithm, and concluding whether it's a good alternative.

General Terms

Algorithm, Analysis.

Keywords

Keyspace, Hashing, bcrypt, SHA-512, Cryptanalysis, Distributed processing, Cloud. Cluster

1. INTRODUCTION

1.1 Motivation

Passwords are the primary method of protection against the cyber-attacks. They are not only used in authenticating an account but also to maintain the privacy of the user , whether it be your email account, e-commerce account , banking account or any other account. Storing passwords in database as clear text is dangerous. The hashing algorithm consists of a hash function which converts the input text called as key into a message digest known as hash. A comparison of the performance of two strong [3] hashing algorithms-SHA -512 and Bcrypt was made, analysing speed of the algorithms over CPU and GPU run on multiple nodes.

1.2 Problem Definition

The main objective of the study is to prove that the time required for computation of hashes on multiple nodes is less as compared to that on a single node. Hence, with the cost remaining constant it is possible to search the keyspace faster by increasing the number of nodes. Since, SHA-512 is an inherently fast algorithm, it is possible to compute more hashes in less time and thus bcrypt could be a feasible alternative to improve security.

2. DESIGN

2.1 Hardware Specification

The Amazon Elastic Compute Cloud (Amazon EC2) service provided by Amazon AWS was used, to deploy the instances which in turn acted as nodes, in the cluster. The reason for choosing EC2 was that it was possible to create custom Linux images, which allowed the node to be up and running with needed dependencies preinstalled. The specification of the instance types are:

C4.8xlarge:

- 36vCPU provided by the Intel Xeon E5-2666 v3.
- 60GB memory.

G2.8xlarge:

- 32vCPU provided by the Intel Xeon E5-2670.
- 60GB memory.
- 4 GRID K520 GPUs:
    - 800 MHz
    - 4GB VRAM
    - CUDA cores: 3072

The c4.8xlarge node was used in CPU clusters, whereas the g2.8xlarge node was used in GPU clusters. To make things more affordable, spot instances were used, which allows bidding on unused instances and generally at 200% lesser cost.

2.2 Algorithms

2.2.1 SHA-512

- SHA-512(Secured Hash Algorithm) is a type of hashing algorithm that operates on eight 64 bit words.
- The first step is called as padding in which 1's and 0's are appended. The next step involves passing through the 80 pre-processing functions in which the various operations like XOR, AND, OR takes place. The third step includes the initialization of eight buffers. The penultimate step involves the processing of the message in 512 bit blocks which consists of predefined and input functions. Ultimately, one receives the output as final message digest in the 8 word buffers.
- It is free of collisions. It computes the fixed length hash irrespective of number of bits in the original text. The maximum message size is 2^{128} -1 bits.
- The working and explanation of SHA-512 is given in [4]

2.2.2 bcrypt

- bcrypt is a type of hashing algorithm based on blowfish that takes passphrases that are 8 to 56 characters long and hashes them internally to a 448 bit key.

- In bcrypt, regular characters are not used. A password X always takes the same amount of time

  regardless of how powerful the hardware is that's used to generate X.
- bcrypt requires you to specify a cost/workfactor in order to generate a password
- This workfactor not only makes the entire process slower but is also used to generate the end hash
- The workfactor allows us to determine how expensive the hash function will be.
- [5] discusses the implementation of bcrypt using special hardware.

2.3 Distributed System Architecture

For distributed processing, one node/PC acts as a master. The master can also act as a salve for computing. But the master itself doesn't have to be computationally fast. Master handles the division of keyspace and distributing workload amongst the slaves.

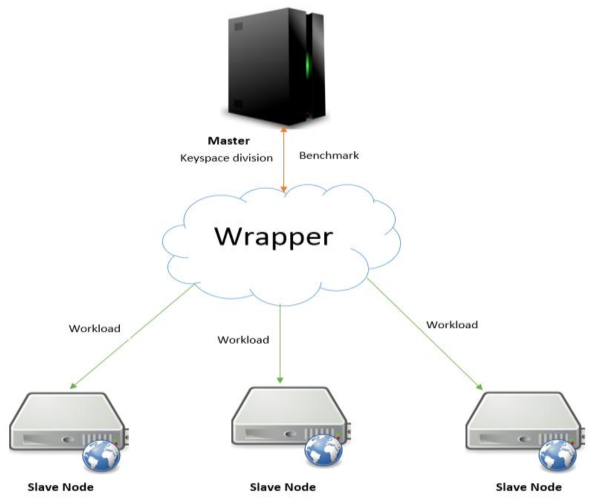


Figure 1. Overview of distributed system architecture

When a slave joins the cluster, its performance is benchmarked for a number of hashing algorithms, and is recorded by the master. As seen in Figure 1, when a hash list is loaded at the master for cryptanalysis, the total keyspace is divided based on the benchmark of the nodes participating in the cluster at that particular time. And the keyspace is not divided completely, but partially for a particular amount of time. Once this time period is over, the next workload, that is part of the keyspace, is requested for further computation. This method reduces network overhead, keeping communication to a minimum of workload distribution, which gives a linear increase in computational speed. Even if a node fails in between, its keyspace workload can be sent to some other node to compensate. Hashcat and cudaHashcat were used on the nodes under the wrapper which handled the computation.

Example:

Workload size: X

Machine can perform these many hashes/second (Benchmark): Y

Total keysize: K

Workload time: (Time after which next workload will be sent to node): Z

X = Y*Z

Assuming 2 nodes,

Keyspace starts from:

0 to N for first node (N<<K)

Therefore next workload is N+1 to N+1+X

2.4 Attack Types

There are many possible attacks used for hash recovery. Some of the common ones are:

2.4.1 *Dictionary attacks*

A word list consisting of many common words, common passwords, and probable passwords is used for hash generation. The advantage of using this is this method is very quick since even the most exhaustive of wordlist contains a billion words. This is the first point of attack. It fails if a more complex or random password is chosen. [6] discusses the time and space complexity involved in Dictionary attacks.

2.4.2 *Rule based attack:*

This acts as an add-on to the dictionary attack. There are a set of rules which are applied to a dictionary to try various combinations of the words present in the list. This results in a larger portion of the probable keyspace being attacked. There have been many successful results using patterns and such rules. [7]

2.4.3 *Rainbow tables:*

Time is lost is computing hashes from plaintext. To save this time, hashes are pre-computed and stored in tables beforehand itself. This speeds up the process since the only computational task is comparing the target hash to the hashes in the rainbow table. These rainbow tables are huge in size, and thus are generally limited to wordlist and rule combinations. The point of failure of this attack is salting. Salting is the process of adding a user specific text to the plain text, before computing the hash. This renders rainbow tables useless since even same passwords will result in different hashes due to salting.

2.4.4 *Bruteforce:*

The focus of the study is on bruteforce attacks since common passwords can easily be found using the above techniques. The last alternative is trying all possible combinations for a password, which is what bruteforce does. But this should be the last resort since keyspace increases exponentially as number of characters increases.

3. PERFORMANCE MEASUREMENT

3.1 Performance with cluster nodes

Table 1. Performance on different number of nodes

	SHA-512	**bcrypt**
Speed	Million Hashes/Second	Hashes/Second
1	280	3310
2	560	6620
3	840	9930
4	1120	13240

It is seen that the performance of the algorithms increases linearly with increase in the number of nodes. The speed of

SHA-512 which is 280 million hashes/second becomes 4 times, i.e. 1120 million hashes/second when 4 nodes are used. This is because the keyspace is divided over the available nodes and the algorithm runs at the same speed at each of the nodes, thus effectively, the total speed of computing is the sum of speeds at each node.

3.1.1 bcrypt

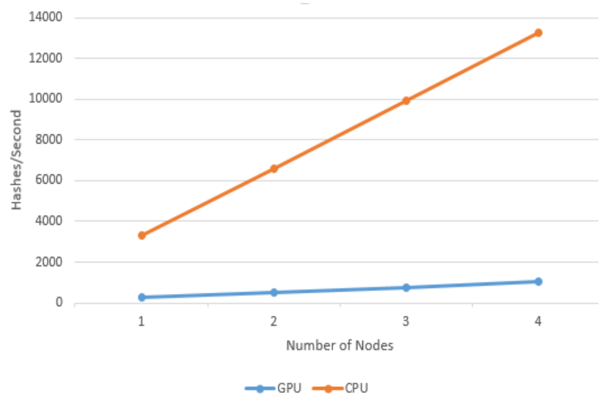


Figure 2. CPU vs GPU

According to the Figure 2, in bcrypt, the performance over CPU is better than the performance over GPU. This is opposite to most of the other algorithms. Thus it is difficult to crack a passphrase encrypted with bcrypt if it is attacked over GPU. Also, the performance is in the order of thousand hashes/second over the CPU which is still considerably lower than the performance of most other algorithms. With GPU never going over 60% utilization, thus was not performing optimally.

3.1.2 SHA-512

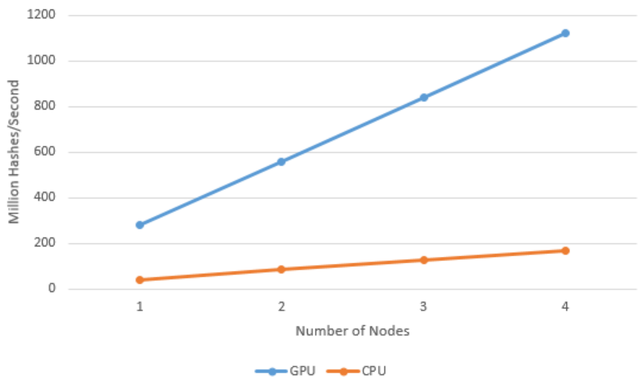
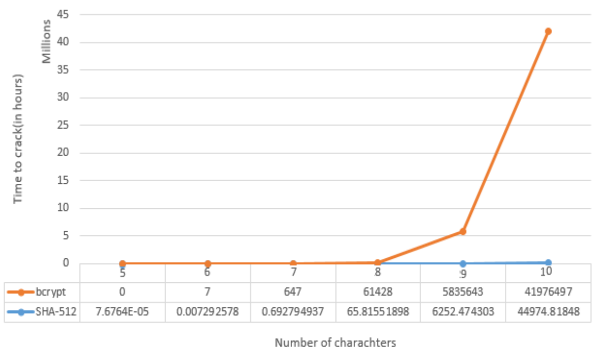


Figure 3. CPU vs GPU

It is clear from Figure 3 that in SHA-512, it is easier to crack the password when deployed over GPU rather than CPU, which is in stark contrast to bcrypt. Also, it's important to note that as the number of nodes increase, more hashes can be cracked in both CPU as well as GPU. Moreover, the performance is in the order of millions of hashes per second which is considerably higher as compared to bcrypt, making SHA-512, easier to crack. This increase in speed is due to the ability of the GPU to perform 32 bit operations much faster than the CPU.

3.2 Keyspace analysis based on time and cost

3.2.1 Time



	5	6	7	8	9	10
bcrypt	0	7	647	61428	5835643	41976497
SHA-512	7.6764E-05	0.007292578	0.692794937	65.81551898	6252.474303	44974.81848

Figure 4. Time vs Number of Characters

The graph in Figure 4 shows the relation between number of characters and time to crack in million hours. As one can see from the above graph, the time to crack remains fairly constant in SHA-512 as the number of characters increase from 5 to 10, however in bcrypt, the time to crack remains almost constant but increases exponentially over 8 characters, making it almost impossible to crack thereafter.

3.2.2 Cost

The cost incurred is dependent on time required to run the instance. As the number of nodes increase, the time required to compute the number of hashes decreases by the same factor, but the cost per node increases linearly. Thus, even though one can compute faster by increasing the nodes, the overall cost remains the same.

3.3 Distributed processing analysis

Scanning a large keyspace using a single machine can be very time consuming, and might not be feasible. But dividing the keyspace into multiple parts allows multiple nodes to traverse parts of the keyspace. Doing this gives the probability that a hash could be found much sooner if it is in the part of the keyspace towards the end. The algorithm proposed is most effective over a large keyspace, since it might be possible that for a smaller keyspace, division will not be feasible and will result in a performance drop versus a single machine.

This method is most effective when a large hashlist is being analyzed, since up to 100 million hashes can be brought into memory, beyond which a performance drop occurs. More the hashes, higher is the probability of a hash being found by dividing the keyspace. This happens due to the fact that passwords are widely distributed along the keyspace, and won't be found in a particular part.

4. bcrypt vs SHA-512

Moore's law states that every two years the amount of transistors that can be put in a computer doubles. So quicker the hardware gets, recovering hashes becomes realistic fast. One just needs to increase hardware. This why SHA-512 is not the best choice to continue with. Even though collisions are nearly impossible to find, today itself it's feasible to recover the average man's password without spending too much of time. Salting may seem to add another level of difficulty, but the answer to that again is, adding more hardware. bcrypt is derived and expanded upon Blowfish cipher. The primary thing that makes it slow is a part reused from Blowfish, an internal RAM table is involved which is altered through most of the steps. This makes parallelization hard, which in turn results in very slow speeds on the GPU, since GPU memory is shared amongst cores, and ends being not as fast as the CPU/RAM combination. Due to this the GPU cannot throttle at its maximum capacity.

Table 2. Time (ms) to compute hashes

bcrypt		SHA-512	
Cost	Time	Iterations	Time
10	0.034	80,000	0.031
11	0.045	160,000	0.043
12	0.060	320,000	0.060
13	0.2	640,000	0.19

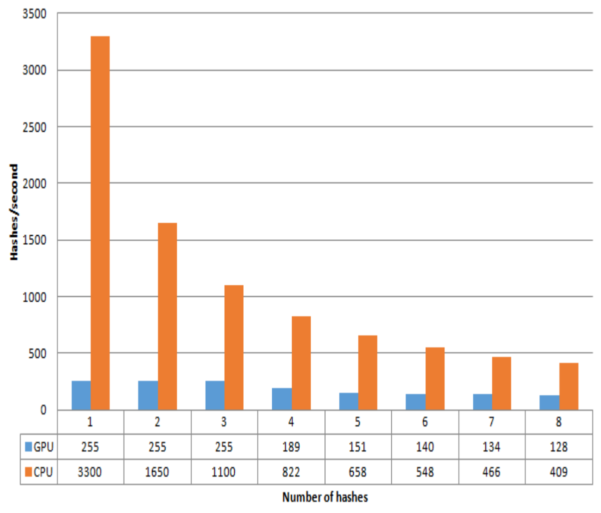
As seen in Table 2, similar speed in computation can be achieved using both algorithms, by adjusting cost factor and iterations. But bcrypt shines due to its ineffectiveness on GPU based attacks. With faster hardware, just increasing the cost factor, makes the hash secure abiding Moore's law [8].

5. CONCLUSION

The primary observation is that, if the keyspace being traversed is divided across nodes, time required reduces sharply at the same cost, if it was run on a single node. Current approaches which create a cluster where the entire cluster acts like a single computational unit, has a tendency to fail if a node fails. The wrapper solution proposed overcomes this without a drop in performance. SHA-512 was designed to be inherently faster, but that is not the most ideal case in real life. bcrypt is slower, but should be adapted more often, since speed is irrelevant if security is being compromised. A 7 character, lower case SHA-512 hash's plaint text can be computed in less than 3 seconds using a 10 node GPU cluster. So it's better if a user has to wait for less than half a second for a bcrypt hash comparison, rather than a millisecond, if higher security can be provided. It was found that the performance decreases when the number of hashes being checked against, in bcrypt was increased. And since the tool used was close source, writing a CPU/GPU based hash computer from scratch that can handle scalability well is the top agenda. The future scope would be to optimize the algorithm to handle different types of hash computations simultaneously if 100% usage is not reached on that particular node. Also trying to optimize GPU performance of bcrypt by trying to better utilize GPU memory in parallel.

6. LIMITATIONS

One of the anomalies found while testing bcrypt was, that the hash calculation speed dropped as the number of hashes being tested against increased. This is shown graphically in Figure 5. Theoretically, number of hashes should not affect the hash computation speed, but this was not the case.



	1	2	3	4	5	6	7	8
🟦 GPU	255	255	255	189	151	140	134	128
🟧 CPU	3300	1650	1100	822	658	548	466	409

Figure 5. Speed vs number of hashes

This might be a limitation of cudaHashcat or hashcat itself, and was beyond the scope since both are closed source applications, whose algorithms weren't available for analysis.

7. REFERENCES

[1] Lin Zhou and Wenbao Han, "A brief implementation analysis of SHA-1 on FPGAs, GPUs and Cell Processors", 2009 International Conference on Engineering Computation, IEEE,Pages 101-104,May 2009

[2] Shay Gueron,et al., "SHA-512/256", 2011 Eighth International Conference on Information Technology: New Generations, IEEE.Pages 354-358 ,April 2011

[3] Kelly Brown, "The Dangers of Weak Hashes", SANS Institute InfoSec Reading Room, June

[4] "Descriptions of SHA-256, SHA-384, and SHA-512"[Online].Available:http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf

[5] Wiemer, F. and Zimmermann, R. 2014 High-speed implementation of bcrypt password search using special-purpose hardware. IEEE conference Publications. Pages

[6] A. Narayanan and V. Shmatikov, "Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff"[Online].Available:http://www.cs.cornell.edu/~shmat/shmat_ccs05pwd.pdf

[7] Tatli,E.I., August 2015. "Cracking more password hashes with patterns." IEEE. Pages 1656 -1665

[8] Katja Malvoni, et.al. "Energy Efficient bcrypt cracking with low-cost parallel hardware",USENIX Association,Berkley,CA,USA,2014