Password encryption using parallel Blowfish algorithm

Marija Anakieva, Vladimir Zdraveski

November 2022

Abstract - In this paper a detailed description of the Blowfish algorithm is given. The basic structure of the algorithm is shown and explained. Block diagrams of the algorithm are presented in order to get the idea. After which a different implementation is proposed-parallel implementation. The time needed for every version of the algorithm to finish is measured and presented in a table. The main focus of the paper is to analyze and compare the different versions of the algorithm on different sizes of data. By comparing all the measured results a conclusion is given.
Key words-encryption, Blowfish algorithm, parallelism

1 Introduction

Many applications require data storage in order to work properly. However this data may be sensitive. In order to ensure privacy, security and data integrity it can't just be saved in the database as it was written. This is where the encryption algorithms come in hand. Before writing in the database and after reading from it sensitive data must be processed and this processing must be fast so that the overall performance of the application isn't affected.

There are several encryption algorithms available, with AES and DES being the most well-known, but in this paper, we will focus on the Blowfish algorithm. Blowfish is an encryption technique designed by Bruce Schneier in 1993 as an alternative to DES encryption. It offers significant advantages over DES, including speed and a high encryption rate, and is free for anyone to use. Despite its speed, there is room for improvement, which is why we propose a new implementation of the Blowfish algorithm - a parallel version.

In the following sections we will be exploring the work that others have done in this field, the basic architecture of the Blowfish algorithm-the key steps needed for transforming sensitive data and possible improvement using the approach of parallelism. In other words we will be comparing the sequential version with the different parallel versions of the algorithm.

2 Related work

Researchers have extensively studied the encryption and decryption algorithms in order to determine their security[6] and evaluate their performance. This has led to a search for a new way of implementing these algorithms. One way that has proved to be successful is by using parallelism.

There are various research articles that focus on the improvement of encryption algorithms using parallel processing. This goes back as far as the 1990s when parallelism was exploited in hardware implementation [3]. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) have the distinction of being introduced in both hardware and software and their parallel hardware implementation has shown to have many advantages, such as increased performance and improved safety[10]. With the improvement of cryptanalysis more and more applications were starting to use Advanced Encryption Standard (AES) instead of Data Encryption Standard (DES) to protect their information security. Because of this a new algorithm for AES parallel encryption was proposed, and designed that implemented a fast data encryption system based on GPU[5].

The basic approach which is identified to be the best and mature method for implementing the same execution process in parallel programming is the predominant "Message Passing Interface (MPI)"[9]. Also in accordance with the OpenMP standard an approach of parallelism based on a spatiotemporal chaotic system and a chaotic neural network has been presented[4].

Various research papers that explore the parallel Blowfish algorithm have also been published. There

are different ways for this to be done for example using parallel computing on GPU [7] or high throughput hardware architecture in RAM blocks [8]. A study was also performed on a supercomputer to evaluate the enhancement of the Blowfish algorithm [2].

In addition to parallelism for data encryption there are also multiple methods of improving image encryption using parallelism. For starters the chaotic encryption based on "Message Passing Interface(MPI)" can be used here as well[1].

3 Solution architecture

All encryption/decryption algorithms follow the pattern shown in Figure 1. where plain text is the data we want to be secure.

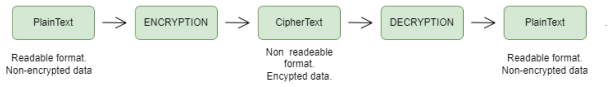


Figure 1. Sequential encryption/decryption diagram

Blowfish is a simple, fast, variable-length symmetric cipher. Which means it uses a single encryption key for both encryption and decryption and the length of the key is chosen by the user according to their needs. The main drawback of this cipher is that it has a 64-bit block cipher which is considered pretty small. However Blowfish has withstood various attacks over the years and many products in many different areas of the Internet still utilize it. One of the biggest usages is for password encryption.
There are two major parts that need to be implemented:

1) Key expansion and subkeys

In the key expansion process, maximum size 448-bit keys are converted into several subkey arrays totaling 4,168 bytes. Subkeys form an integral part of the Blowfish algorithm, which uses a large number of them. These subkeys are pre-computed before encryption or decryption can take place.

2) Data encryption

Data encryption happens through a 16-round Feistel network, with each round consisting of a key-dependent permutation and a key- and data-dependent substitution. Large, key-dependent S-boxes work with the substitution method and form an integral part of the data encryption system in Blowfish. All encryption operations are XORs and additions on 32-bit words.

For example if we have a password written as "123456abcd132536" and we ran the Blowfish algorithm on it the password would go through 16 rounds each round transforming it to something completely different and at the end instead of the original text "d748ec383d3405f7" would be stored in the database.

On Figure 2 a block diagram of all the operations that the plain text goes through in just one round is shown.

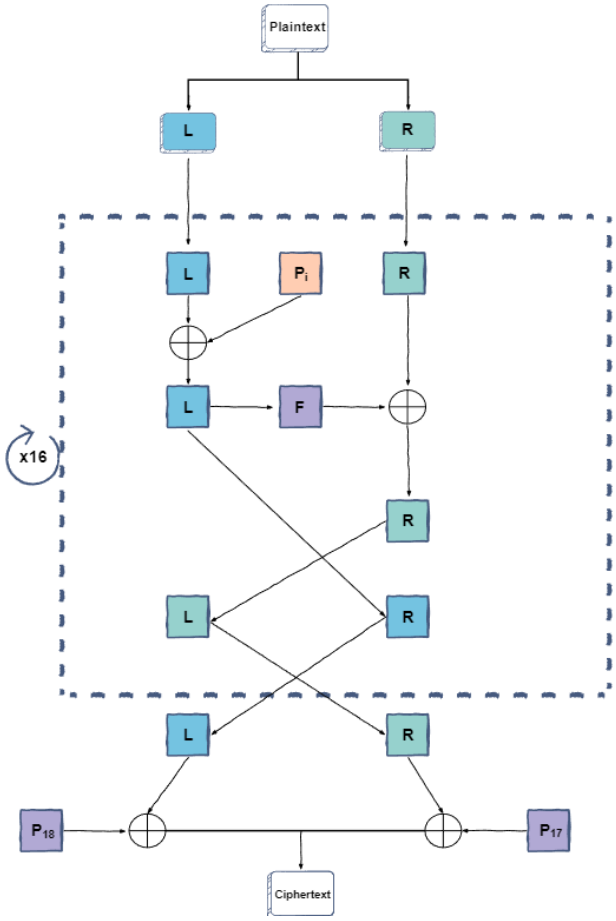


Figure 2. One iteration

The process of generating subkeys could be a pretty slow process but it only happens ones so this isn't the part we want to improve. In order to improve the overall performance we want to parallelise the second

major step-the encryption. This will be achieved by dividing the data that is supposed to be encrypted into a few subsets, so that each subset can go through the 16 Feistel-like iterations at the same time. This way instead of each password waiting for the previous password to be encrypted one password from each subset would be going though the process at all times. The number of passwords that are being encrypted simultaneously will depend on the number of subsets.

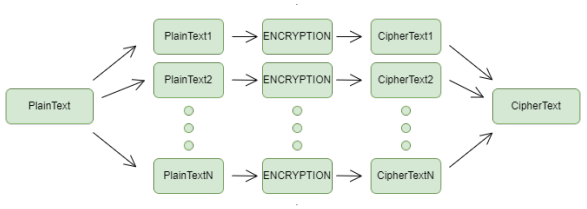


Figure 3. Parallel encryption diagram

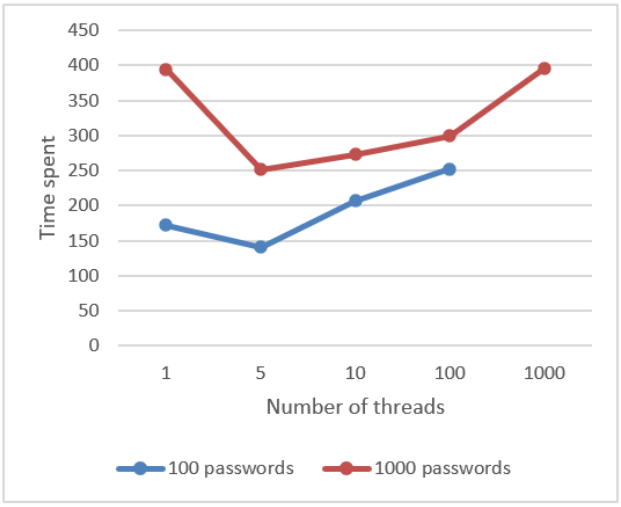
4 Results

After the algorithm was implemented it was ran on different amounts of data: 100, 1000, 10000, 100000 and 1000000 passwords and the time needed for each set of passwords to be encrypted was measured. Besides changing the amount of data we also changed the number of threads in order to determine which number of threads gives us the best results for the specific set.

Number of threads	Number of passwords: 100	1 000	10 000	100 000	1 000 000
1	172	394	1 509	10 309	81 915
10	207	273	672	3 818	34 293
100	252	299	808	4 197	33 379
1 000	/	396	1 552	4 488	36 639

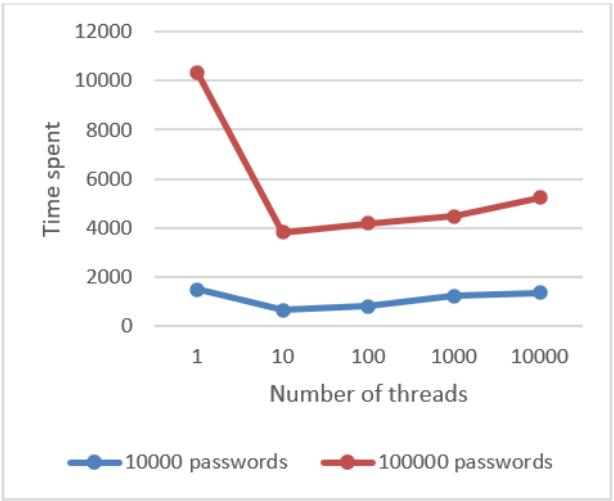
Table 1. Average time

In Table 1 we can see the average time needed for the algorithm to finish for each set and each number of threads. The time is expressed in milliseconds. The first row of the table (1 thread) shows us the time needed if the algorithm runs sequentially.

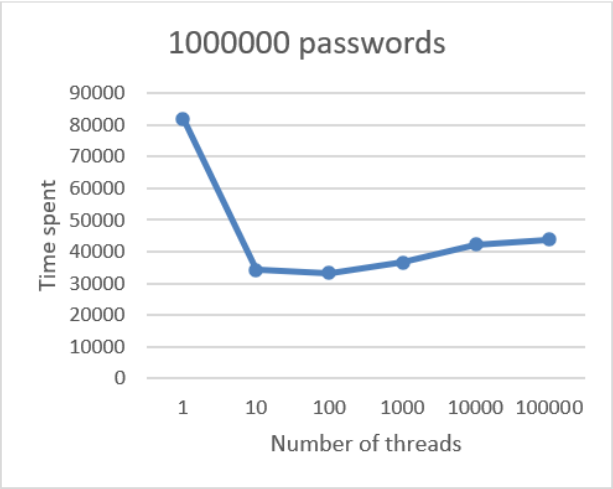


Graph 1. Small sets

As we can see on Graph 1 for small sets of data for example 100 and 1000 passwords there is only a small decrease in execution time after the parallelism of the algorithm. However if we use too many threads the algorithm becomes even slower than the sequential one. Because of this we can come to the conclusion that for small sets of passwords going through the process of parallelism is not worth it.



Graph 2. Medium sets



Graph 3. Large set

With bigger data sets the drop is much more noticeable. For example when there are 100000 passwords to encrypt the sequential algorithm will take 10 seconds and the fastest parallel algorithm will need only 3 seconds, for 1 million passwords the sequential algorithm will need 1 minute and 20 seconds and the fastest parallel algorithm will be 33 seconds. All of this is considered after measuring the time more times and then calculating the average.

For each set the fastest time is achieved for a different number of threads. We also noticed that usually this number isn't very big and when the number grows the algorithm starts getting slower. By looking at Graph 1, Graph 2 and Graph 3 we can see that this is true no matter the size of the set.

5 Conclusion

After conducting an analysis of the results obtained from measuring the execution time for varying amounts of data and a range of thread numbers, we have concluded that the execution time is directly impacted by both of these factors. The optimal execution time is typically achieved with a larger number of threads, but definitely not the largest number possible. In the end with the modification of the Blowfish algorithm that allowed it to work in parallel we were able to increase its speed measured by hundreds of milliseconds and even seconds with bigger data sets. We can clearly see that there are more advantages of using the parallel algorithm when working with larger datasets as opposed to smaller sets.

In the future this algorithm can be improved by implementing a function that will calculate the optimal number of threads based on the data size.

It is also worth mentioning that the decryption algorithm which is similar to that of encryption (only the subkeys are used in reverse) can also be improved by the same method of parallelism.

References

[1] Mohammed Abutaha, Islam Amar, and Salman AlQahtani. Parallel and practical approach of efficient image chaotic encryption based on message passing interface (mpi). *Entropy*, 24(4), 2022.

[2] Mahmoud Rahallah Asassfeh, Mohammad Qatawneh, and Feras Mohamed AL-Azzeh. Performance evaluation of blowfish algorithm on supercomputer iman1. *International Journal of Computer Networks & Communications (IJCNC)*, 10(2), 2018.

[3] Albert G Broscius and Jonathan M Smith. Exploiting parallelism in hardware implementation of the des. In *Annual International Cryptology Conference*, pages 367–376. Springer, 1991.

[4] Dariusz Burak. Parallelization of an encryption algorithm based on a spatiotemporal chaotic system and a chaotic neural network. *Procedia Computer Science*, 51:2888–2892, 2015. International Conference On Computational Science, ICCS 2015.

[5] Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang, and Conglan Lu. Parallel aes algorithm for fast data encryption on gpu. In *2010 2nd international conference on computer engineering and technology*, volume 6, pages V6–1. IEEE, 2010.

[6] Prerna Mahajan and Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 2013.

[7] Tejal Mahajan and Shraddha Masih. Enhancing blowfish file encryption algorithm through parallel computing on gpu. In *2015 International Conference on Computer, Communication and Control (IC4)*, pages 1–4. IEEE, 2015.

[8] Soufiane Oukili and Seddik Bri. High throughput parallel implementation of blowfish algorithm. *Applied Mathematics & Information Sciences*, 10(6):2087–2092, 2016.

[9] Sampath Kumar Tallapally and B. Manjula. Suitable encrypting algorithms in parallel processing for improved efficiency. *IOP Conference Series: Materials Science and Engineering*, 981(2):022017, dec 2020.

[10] Abdulmajeed Adil Yazdeen, Subhi RM Zeebaree, Mohammed Mohammed Sadeeq, Shakir Fattah Kak, Omar M Ahmed, and Rizgar R Zebari. Fpga implementations for data encryption and decryption via concurrent and parallel computation: A review. *Qubahan Academic Journal*, 1(2):8–16, 2021.