Data Encryption Standard Algorithm

The Program of Software Engineering, ‘Afeka’

Course: Programming languages Seminar, 10356
Lecturer: Dr.Aviv Itzhak
Semester: Spring 2020

Project: Research and Development Python Multithreading - Data Encryption Standard Algorithm

Team Members:

1. Viatcheslav Kagan
2. Liad Khamdadash

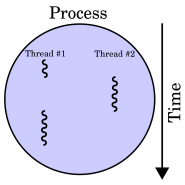


Table of Contents

1. Introduction

Our project Implements Data Encryption Standard algorithm while using python multithreading technique.

1.1 What is Cryptography?

Cryptography involves creating written or generated codes that allow information to be kept secret. Information security uses cryptography on several levels. The information cannot be read without a key to decrypt it and maintains its integrity during transit and while being stored.

1.2 What is Python?

Python is an interpreted language (execute instructions directly without compiling a program into machine-language instructions) high-level, general-purpose programming language. first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs, and object-oriented approach, aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected.

1.3 What is Thread?

A thread is a path that is followed during a program's execution.
Many programs run as a single thread. For example, a program is not capable of reading keystrokes while making drawings. These tasks cannot be executed by the program at the same time. This problem can be solved through multitasking so that two or more tasks can be executed simultaneously.

1.4 What is Multithreading?

Multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to provide multiple threads of execution concurrently, supported by the operating system. This approach differs from multiprocessing. In a multithreaded application, the threads share the resources of a single or multiple cores, which include the computing units, the CPU caches, and the translation lookaside buffer (TLB).

1.5 What is Multithreading in Python?

Python, in general, works slow on a machine.
The default implementation of Python 'CPython' uses GIL (Global Interpreter Lock) to execute exactly one thread at the same time, even if run on a multi-core processor as GIL works only on one core regardless of the number of cores present in the machine.

Multithreading does not make much of a difference in execution time as it uses the same memory space and a single GIL, **so any CPU-bound tasks do not have an impact on the performance of the multi-threaded programs** as the lock is shared between threads in the same core and only one thread is executed while they are waiting for other tasks to finish processing. Also, threads use the same memory so precautions have to be taken or two threads will write to the same memory at the same time. This is the reason why the global interpreter lock is required.

1.6 What is a fork in parallel implementation?

The system function call fork() creates a copy of the process, which has called it. This copy runs as a child process of the calling process. The child process gets the data and the code of the parent process. The child process receives a process number (PID, Process IDentifier) of its own from the operating system. The child process runs as an independent instance, this means independent of a parent process. With the return value of fork(), we can decide in which process we are: 0 means that we are in the child process while a positive return value means that we are in the parent process. A negative return value means that an error occurred while trying to fork.

1.7 What is a thread pool in parallel implementation?

A process pool object which controls a pool of worker processes to which jobs can be submitted. It supports asynchronous results with timeouts and callbacks and has a parallel map implementation. processes are the number of worker processes to use. The threads are the number of threads that you want and tasks are a list of tasks that most map to the service.

2. Pseudocode

```
Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64]) {
    permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
    split (64, 32, inBlock, leftBlock, rightBlock)
    for (round = 1 to 16) {
            mixer (leftBlock, rightBlock, RoundKeys[round])
            if (round!=16) swapper (leftBlock, rightBlock)
    }
    combine (32, 64, leftBlock, rightBlock, outBlock)
    permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}

mixer (leftBlock[48], rightBlock[48], RoundKey[48]) {
    copy (32, rightBlock, T1)
    function (T1, RoundKey, T2)
    exclusiveOr (32, leftBlock, T2, T3)
    copy (32, T3, rightBlock)
}

swapper (leftBlock[32], rigthBlock[32]) {
    copy (32, leftBlock, T)
    copy (32, rightBlock, leftBlock)
    copy (32, T, rightBlock)
}

function (inBlock[32], RoundKey[48], outBlock[32]) {
    permute (32, 48, inBlock, T1, ExpansionPermutationTable)
    exclusiveOr (48, T1, RoundKey, T2)
    substitute (T2, T3, SubstituteTables)
    permute (32, 32, T3, outBlock, StraightPermutationTable)
}

substitute (inBlock[32], outBlock[48], SubstitutionTables[8, 4, 16]) {
   for (i = 1 to 8) {
     row ← 2 x inBlock[i x 6 + 1] + inBlock [i x 6 + 6]
     col ← 8 x inBlock[i x 6 + 2] + 4 x inBlock[i x 6 + 3] + 2 x inBlock[i x 6 + 4] + inBlock[i x 6 + 5]
     value = SubstitutionTables [i][row][col]
     outBlock[[i x 4 + 1] ← value / 8; value ← value mod 8
     outBlock[[i x 4 + 2] ← value / 4; value ← value mod 4
     outBlock[[i x 4 + 3] ←  value / 2; value ← value mod 2
     outBlock[[i x 4 + 4] ← value
    }
}
```

3. Algorithm explanations

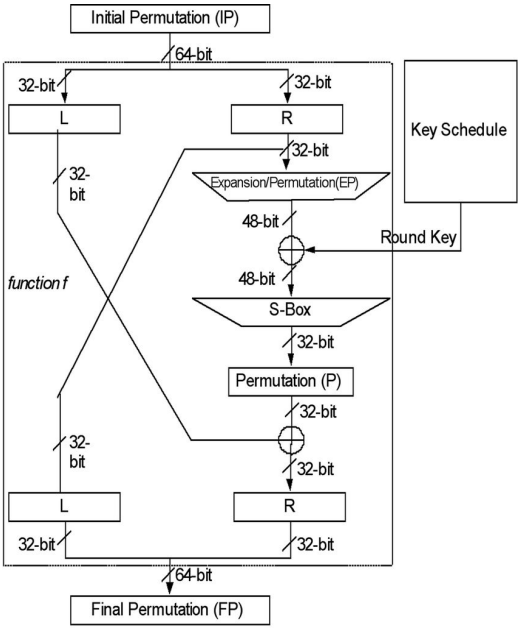
3.1 General

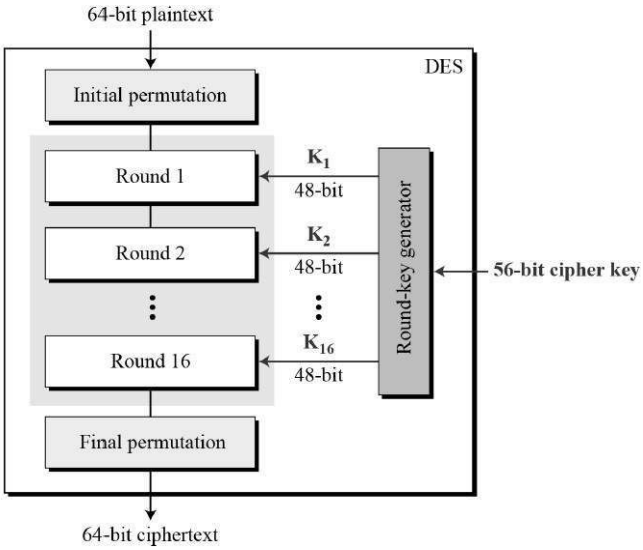
DES is a block cipher algorithm that takes the plain text in blocks of 64 bits and converts them to ciphertext using keys of 48 bits. It is a symmetric key algorithm, which means that the same key is used for encrypting and decrypting data. Both the sender and the receiver must know and use the same private key. Encryption and decryption using the DES algorithm. It is the most popular security **algorithm** and has been found vulnerable against very powerful attacks and therefore, the popularity has been found slightly on the decline.

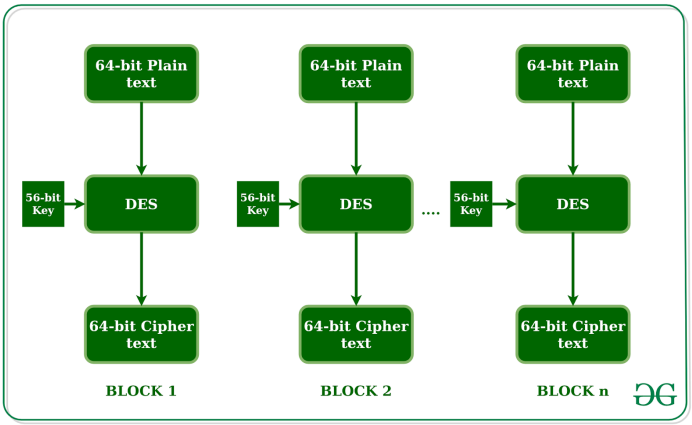
For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the ciphertext "0000000000000000". If the ciphertext is decrypted with the same secret DES key "0E329232EA6D0D73", the result is the original plaintext "8787878787878787". DES is based on the two fundamental attributes of cryptography: substitution (also called as confusion) and transposition (also called as diffusion). DES consists of 16 steps, each of which is called around. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

1. In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function
2. The initial permutation performed on plain text
3. Next, the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT)
4. Now each LPT and RPT to go through 16 rounds of the encryption process
5. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
6. The result of this process produces 64-bit ciphertext

3.2 How it really looks like







4. Code explanations

4.1 General files

textToEnc.py

A long text (a famous story that is taken from the internet) that we want to encrypt using the des algorithm and compare sequential implementation and parallel implementation.

4.2 Sequential section

sequential_des.py

Sequential programming is when the algorithm to be solved consists of operations one after the other and you do not have to do alternative operations. In our code, we just implement the whole algorithm we explained before using all the blocks and tables for that.

4.3 Parallel section

4.3.1 Thread fork implementation

parallel_des_fork.py

```
def run(self, text1: str, action: Cryptography, num_of_threads: int):
    chunks, chunk_size, num_of_times = len(text1), 8, math.ceil(len(text1) / 8)
    str_list = [text1[i:i + chunk_size] for i in range(0, chunks, chunk_size)]
    threads = []
    times = min(num_of_threads, num_of_times)
    for i in range(times):
        t = threading.Thread(target=self.run_block, args=(str_list, i, action,))
        threads.append(t)
        t.start()                    t.join()
    while times < num_of_times:
        for t in threads:
            if not t.is_alive() and times < num_of_times:
                t = threading.Thread(target=self.run_block, args=(str_list, times, action))
                    t.start()
                    t.join()
                    times += 1
```

* **start()** - start the thread's activity. It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.
* **join()** - Wait until the thread terminates. This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception – or until the optional timeout occurs.
* **threading.Thread()** - class threading.Thread(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None)

This constructor should always be called with keyword arguments.
Arguments are:

- group should be None; reserved for future extension when a ThreadGroup class is implemented.
- target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.
- name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.
- args is the argument tuple for the target invocation. Defaults to ().
- kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

4.3.2 Thread pool implementation

parallel_des_fork.py

```
def run(self, text1: str, action: Cryptography):
    chunks, chunk_size = len(text1), 8
    return "".join(ThreadPool().map(lambda s: self.run_block(s, action),
                    [text1[i:i + chunk_size] for i in range(0, chunks, 8)]))
```

* **ThreadPool().map** - The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism)

5. Python Built-in functions used

Annotations-

1. @classmethod - returns a class method for the given function, is a method that is bound to a class rather than its object.
2. @staticmethod - returns a static method for a given function, methods that are bound to a class rather than its object

General functions-

3. Self - represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python.
4. Cls - if the method is a class method then the keyword cls must be used
5. print() - prints the specified message to the screen, or another standard output device
6. list() - constructor returns a list in python
7. math.ceil() - returns the smallest integral value greater than the number

Casting/Converting functions-

8. int() - returns an integer object from any number or string
9. chr() - returns a character (a string) from an integer (represents Unicode code point of the character)
10. str() - returns the string version of the given object
11. float() - returns a floating-point number from a number or a string
12. ord() - returns an integer representing the Unicode character
13. bin() - converts and returns the binary equivalent string of a given integer
14. round() - floating-point number rounded to the specified number of decimals

Object functions-

15. range() - returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number
16. isinstance() - checks if the object (first argument) is an instance or subclass of class info class (second argument)
17. len() - returns the number of items (length) in an object
18. find() - finds the first occurrence of the specified value
19. zip() - takes iterables (can be zero), aggregates them in a tuple, and return it
20. join() - takes all items in an iterable and joins them into one string
21. min() - returns the item with the lowest value or the item with the lowest value in an iterable.

22. raise Exception() - allows forcing a specified exception to occur
23. map() - applies a given function to each item of an iterable (list, tuple...) and returns a list of the results

Time functions-

24. time() - returns the time as a floating-point number expressed in seconds since the epoch, in UTC

File functions-

25. open("guru99.txt", "w")) - opens a file, and returns it as a file object
26. file.write()- "w" - write overwrite any existing content

6. Results and conclusions

In most cases and in other programming languages multithreading is a good way to implement parallel code and it has many advantages comparing sequential implementation.

- It Reduces the time consumption/ response time
- Improving performance and faster access
- Simplifies the code
- Better utilization of resources
- Allows concurrent and parallel occurrence of various tasks

In python, multithreading is **<u>not effective</u>** and most of the cases running time of a project are worse than the sequential. Using the pool implementation is a little bit better than the fork implementation, but it still is not enough to beat the sequential implementation in running time.

Time results depend on the load on the computer.
There are many other processes and so there may be many delays

<u>That is what we got in our project</u> during our research as seen in the images below:

<u>Time results encryption:</u>

encryption_time.txt - Notepad

File Edit Format View Help

```
########### Encryption Time Results ###############
Note: Time results depends on the load on the computer.
 There are many other processes and so there may be many delays

Number of processors in this computer: 8
The results will be from 2 threads to 16 threads

Number of threads              Fork join run time             Data the same as in sequential?
        2                                  12.358 sec                              True
        3                                  12.304 sec                              True
        4                                  12.303 sec                              True
        5                                  12.298 sec                              True
        6                                  12.345 sec                              True
        7                                  12.374 sec                              True
        8                                  12.274 sec                              True
        9                                  12.256 sec                              True
        10                                 12.256 sec                              True
        11                                 12.255 sec                              True
        12                                 12.305 sec                              True
        13                                 12.244 sec                              True
        14                                 12.229 sec                              True
        15                                 12.245 sec                              True
        16                                 12.312 sec                              True
Runtime Parallel Thread Pool result: 8.742 sec
Run time without multithreading: 8.323 sec
```

Time results decryption:

decryption_time.txt - Notepad

File Edit Format View Help

```
########### Decryption Time Results ###############
Note: Time results depends on the load on the computer.
 There are many other processes and so there may be many delays

Number of processors in this computer: 8
The results will be from 2 threads to 16 threads

Number of threads              Fork join run time             Data the same as in sequential?
        2                                   12.303 sec                              True
        3                                   12.235 sec                              True
        4                                   12.306 sec                              True
        5                                   24.491 sec                              True
        6                                   25.219 sec                              True
        7                                   19.706 sec                              True
        8                                   16.111 sec                              True
        9                                   12.347 sec                              True
        10                                  12.300 sec                              True
        11                                  12.238 sec                              True
        12                                  12.306 sec                              True
        13                                  12.340 sec                              True
        14                                  12.276 sec                              True
        15                                  12.274 sec                              True
        16                                  12.251 sec                              True
Runtime Parallel Thread pool result: 8.578 sec
Run time without multithreading: 8.335 sec
```

7. Appendix: Installation and running instructions

1. **Download Pycharm IDE** from the internet and **Install it**
2. **Download the latest python version (3+)** in order to use the language and install it
3. Open the **project directory** from your computer
4. **Install the plugins** that IDE is asking you inside the project
5. Open the **main.py** file and run it (wait for some time to finish)
6. Now you have the **results** in the console itself
7. You have also **4 files that added to the project** after running the program:
    a. **data_results_fork.txt** for the text, the key, and the encryption data in sequential option and parallel option while using threads and fork
    b. **data_results_pool.txt** for the text, the key, and the encryption data in sequential option and parallel option while using a thread pool
    c. **decryption_time.txt** for calculating the decryption time of the text and the time difference between pool, fork and sequential implementation
    d. **encryption_time.txt** for calculating the encryption time of the text and the time difference between pool, fork and sequential implementation

Enjoy using our project!