Distributed Decipher

Aleksandar Kanchev F80806

Programming Practice NETB380

June 27, 2017

Introduction

This paper presents the documentation for one of the submodules of project variant 4 – Distributed Decipher. This project was developed by Team 3, which consisted of Vladimir Petkov (team leader), Zdravko Donev and myself.

The following chapters contain the project requirements, as well as a comprehensive and detailed description of Deciphy – a distributed decipher.

The project source code can be found on GitHub.

Requirements

This section contains the project requirements. It is divided into two subsections. The first one describes the overall project requirements – those are shared amongst all teams. The other one lists the project-specific requirements that the team has come up with when initially defining the boundaries of this project.

Overall requirements

The overall requirements of the course were to create a program written in the C++ programming language using the Qt framework. The program needed to have a GUI (graphical user interface) and have one of the following additional features:

- performs simple processing of image files;
- stores and reads data from a relational database;
- performs a connection over the Internet or a local network.

Project-specific requirements

The specific goal of this project was to create a program that could successfully decipher English text, deciphered using the Vigenere cipher. The desired workflow was simple – user enters encrypted text, clicks a button and sees the original text.

Vigenere cipher

Since this project allows for decrypting Vigenere-encrypted texts, this section provides some background information about the cipher in question.

History

The Vigenere cipher is based on the well-known Caesar cipher. However, it uses a series of interwoven Caesar ciphers based on the letters of a keyword. Thus, it is much more complex and hard to break than its predecessor.

It was originally described in 1553 by Giovan Battista Bellaso. Despite of that, it is wrongly attributed to Blaise de Vigenere, who only added slight improvements to the cipher in 1586.

This cipher is also known as the “the indecipherable cipher” because it stayed impenetrable for 3 centuries. It was even used during the American Civil War. By this time, it had already been cracked and was only used to send non-critical messages.

Mechanics

This subsection provides details about the mechanics behind the Vigenere cipher.

As mentioned previously, this cipher uses a series of Caesar cipher transformations. These are reflected in a matrix, used when encrypting texts with the Vigenere cipher. The matrix contains all letters of the desired alphabet in its rows and columns. However, the letters are shifted by one position in each subsequent row. Such a matrix is shown in Figure 1.

To encrypt a text using the Vigenere cipher, the sending party chooses a key that they can share with the receiving party through a secure channel. Then, the key is repeated as many times as needed to get the same number of letters as contained in the text to be encrypted. Lastly, for every position in the text, the sender uses the matrix to lookup the letter that is at the intersection of the corresponding letters from the repeated key (in horizontal) and the original text (in vertical) at the given position.

Figure 1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Example:

Plaintext: **QT IS AWESOME**
Key: **CPP**
Ciphertext: **SI XU PLGHDOT**

Solution

This section provides a detailed description of the implemented solution.

Method

Even though the information on how to crack the Vigenere cipher has been available to mankind for couple of centuries now, it still requires a very special set of cryptanalysis skills to decrypt a message encrypted with it. Thus, this solution relies

more on exhausting all possibilities, rather than a clever decryption technique. As such, this deciphering software falls into the brute-force category.

The brute-forcing approach is used when guessing the key that was used to encrypt the text. Without the original key, the secret text remains a mystery. The more actual English words we get when decrypting the crypto text with a given key, the higher the probability that this is in fact the original key.

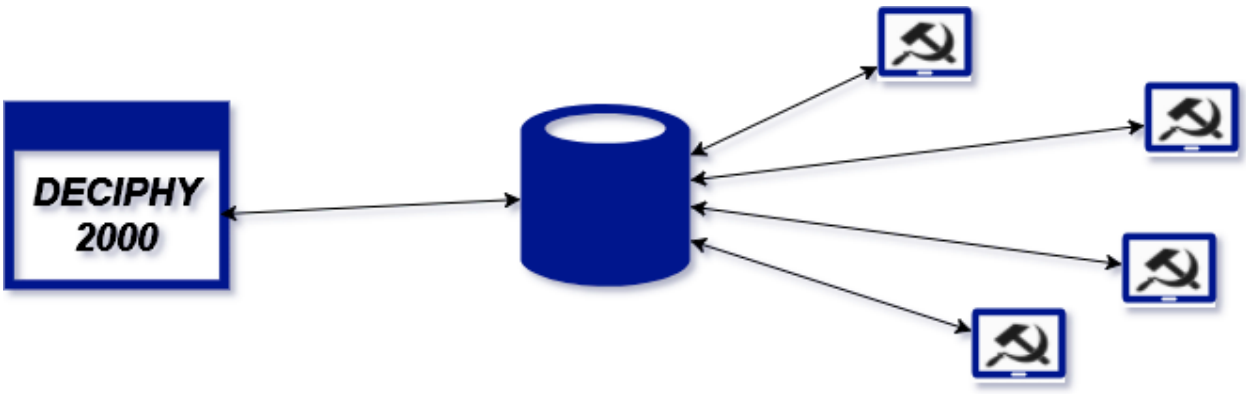
High-level architecture

To make this computer program more performant and fault tolerant, the work of decrypting the provided text is parallelized. Instead of having one instance of the application do all the work, the application runs on several nodes each working on solving only part of the whole problem. This idea involves having a master node which shows the UI and divides the problem into chunks. Each chunk represents a self-contained task that can be executed individually and in parallel to other tasks. In addition to the master, this architecture requires someone to execute the tasks. This is handled by the slave (worker) daemons. They are a collection of identical lightweight console applications. The worker daemon is described in more details in the Worker daemon section of this paper.

The way the master communicates tasks to the slaves is through a database. Once, the master divides the problem into numerous small problems, it submits them to the database. Then, any slave that is currently unoccupied, will pick a task and execute it.

Figure 2 depicts the architecture described above.

Figure 2



One of the main requirements for this project was that it should perform communication to a remote host over the Internet. The architecture described

above also allows for the master, slaves and DB to all live on separate physical machines.

Database schema

As per the requirements, this project uses a relational database. Postgres SQL has been chosen, since the default Qt installation provides drivers for accessing it.

This project uses a single database, called *deciphy*. It contains two tables – *texts* and *tasks*. The *texts* table stores all texts that have ever been inputted into the application. The *tasks* table, on the other hand, can be referred to as the workers' activity log.

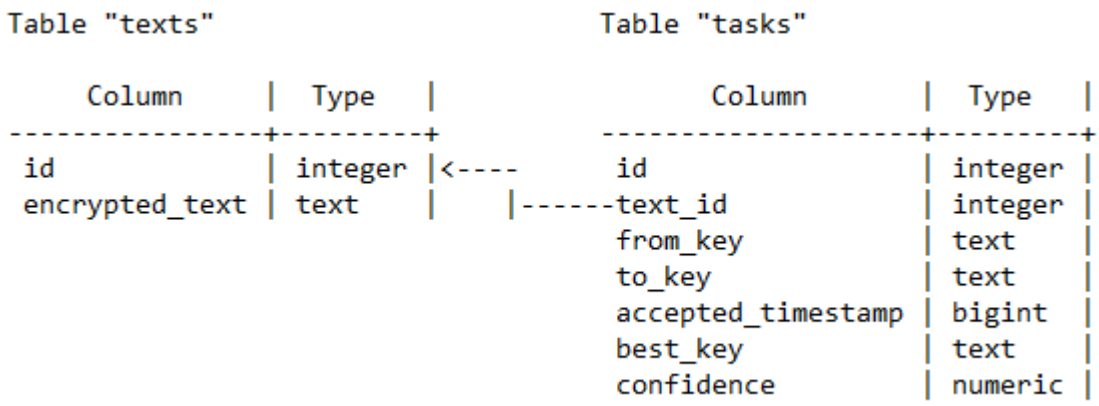
Each entry in the *texts* table contains two columns – *id* and *encrypted_text*. The *id* column is the primary key for this table. The *encrypted_text* represents the text to decrypt.

The entries in the *tasks* table are represented by the following columns:

- *id* – primary key of the table
- *text_id* – a foreign key, pointing to an *id* in the *texts* table
- *from_key* – left boundary in the key-guessing range
- *to_key* – right boundary in the key-guessing range
- *accepted_timestamp* – the time when this task was started. Allows other workers to know if the task has already been started by someone else. Also, allows for implementing fault tolerance. For example, if a task has not been completed for more than 2 minutes, other workers are free to pick it up.
- *best_key* – this is one of the two results from a successfully executed task. It represents the key that showed best results in decrypting the text from all keys within the given range.
- *Confidence* – this is the second result from a successful task execution. It is a numeric floating-point number between 0 and 1. It represents the percentage of words in the text, decrypted using the *best_key,* that exist in the dictionary used by this application.

The following figure illustrates the structure of the two tables and the relation between them.

Figure 3



Worker daemon

NOTE: This section uses the terms *worker, daemon, worker daemon* and *slave* interchangeably to refer to one and the same thing.

As state above, the worker daemon is a lightweight console application. Its sole purpose is to process tasks from the *tasks* table in the database. If no unprocessed tasks exist, the worker will sleep for 30 seconds.

Once a slave finishes a task, it submits the result to the database. The result is defined by the two properties *best_key* and *confidence* described above. The implementation of this deciphering software allows for an unlimited number of workers to be added. The more slaves, the faster the decryption.

Last, working in a distributed environment requires handling cases where one slave dies in the middle of executing a task or additional slaves join in the middle of the decryption process.

This application takes the above problems into account. It implements the following mechanisms to prevent from getting into an inconsistent state because of the changing slave landscape:

- The master and slave nodes do not know about each other. More importantly, the master does not keep track of all available slaves. Instead, the slave nodes proactively take tasks and execute them. The master only queries the database on a given time interval to find out, if a key with a high-enough confidence has been found.
- If a slave does not complete a task within a given time interval, the task will be picked by another slave. This mechanism considers the average task

execution time and assumes that something has went wrong if the task execution is prolonged far more than this interval.

- Tasks are independent. This also contributes to this solution being agnostic to any failures in the slaves as no node requires input from the rest. Thus, the most critical point in this application is the database, as it keeps all the information needed for a normal program execution.

References

Vigenere Cipher. (2017, May 12). Retrieved from Wikipedia, the free encyclopedia:

https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher