Developer Test

Introduction

This test has the purpose of assessing your proficiency on a programming language, your capacity to understand technical documents in English and your ability to learn new skills and overcome challenges.

Overview

To pass this test, you must write a computer program in any language of your choosing that shall:

1. Establish a TCP connection with our server
2. Receive a message encoded with "T-Scheme"
3. Validate the received message according to "T-Scheme"
4. Break the cipher contained in the message
5. Encode the revealed plaintext with "T-Scheme"
6. Send response to server
7. Receive, validate and decrypt the confirmation message from server according to "T-Scheme" and the key used previously to encrypt the first message

Enroll for the test

Access the website at the following URL:

http://191.237.249.140:8000/

There you will be prompted for your name and email address. Once you have entered the requested data, you will be informed of the communication parameters you should use to connect with the server (host IP and port). You will also be able to view the packets sent and received between the server and your program.

T-Scheme

The T-Scheme is a fictional protocol devised solely for the purposes of this test. It defines a message format for encoding and decoding messages between the server and the client, and also a sequence of messages that comprises the communication between server and client.

Message format

The T-Scheme divides all messages in 4 parts:

Length	md5	data	parity
2 bytes	16 bytes	n bytes	1 byte

Length: Is the first 2 bytes of the packet. The length is stored as a big endian unsigned short int. It contains the whole message length in bytes, including itself.

Md5: The md5 hash value of data.

Data: The actual bytes being transmitted.

Parity: A byte filled so as to maintain the whole message with an even number of 1-bits.

Communication sequence

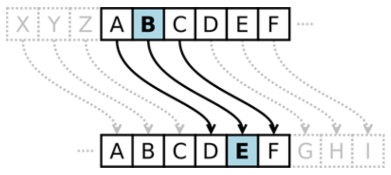
The communication between the server and the client shall follow the following sequence:

1. The server sends a message to the client containing as data the encryption of an ASCII-encoded plaintext
2. The client sends back to the server a message containing as data the ASCII-encoded decrypted plaintext
3. The server sends a message to the client containing as data either:
    a. The encryption of the ASCII-encoded string “OK” using the same key as the first message; or
    b. The string “Wrong”, ASCII-encoded

The encryption done by the server shall be a Monoalphabetic cipher, as described in the next topic. The client shall break the cipher after receiving the first message in order to send back to the server the decrypted plaintext.

Monoalphabetic Cipher

A monoalfabetic cipher is a substitution cipher that uses fixed substitution over the entire message. One of the classic substitution cipher is the Caesar Cipher. It is said that the emperor Julius Caesar, in the early steps of cryptography, used to communicate with the roman army generals by shifting the letters 3 times to the right on the alphabet.



Thus, the encryption of HELLOWORLD using the Caesar Cipher should be: KHOORZRUOG. To decrypt, you simply shift back 3 letters. We can say that the key is the number 3, because was the number used to initially shift the alphabet.

In this test we are going to use the same idea, but instead of shifting letters we are going to operate their ASCII value.

First we choose a key at random: 0x7B

Now, we compute the XOR value between every byte of the ASCII encoded message:

Plaintext ASCII	H	E	L	L	O	W	O	R	L	D	
Plaintext bytes	48	45	4C	4C	4F	57	4F	52	4C	44	
key	7B	7B	7B	7B	7B	7B	7B	7B	7B	7B	⊕
Ciphertext bytes	33	3E	37	37	34	2C	34	29	37	3F	
Ciphertext ASCII	3	>	7	7	4	,	4	)	7	?	

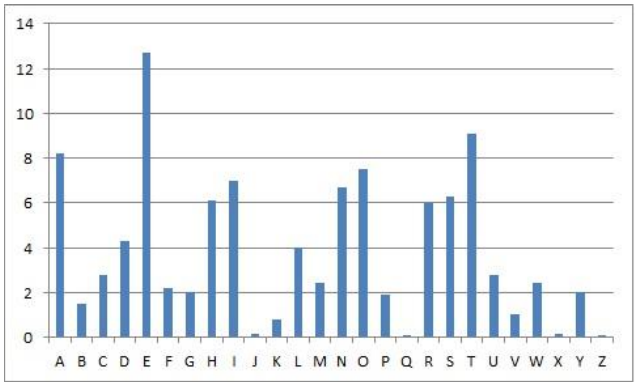
To decrypt the ciphertext you simply compute XOR with the key for each byte of the ciphertext, yielding the plaintext again.

Ciphertext ASCII	3	>	7	7	4	,	4	)	7	?	
Ciphertext bytes	33	3E	37	37	34	2C	34	29	37	3F	
key	7B	7B	7B	7B	7B	7B	7B	7B	7B	7B	⊕
Plaintext bytes	48	45	4C	4C	4F	57	4F	52	4C	44	
Plaintext ASCII	H	E	L	L	O	W	O	R	L	D	

We can notice that the letters were changed, but not their order or their distribution in the text. This information is extremely relevant for the next topic.

Frequency Analysis

The frequency analysis turned out to be a powerful method for breaking all substitution ciphers. Each letter in a text book has not only a singular distribution but also can give a hint about the following letter. The image below shows the letters distribution in the English Language.



It can be seen that the letter "e" is the most common letter in a regular English text. Since the frequency of the letters are not changed in the ciphertext, the most common letter in the cipher might be the encryption of the letter "e", so operating these two letters, we can obtain information about the key used in the encryption process:

"e" ⊕ key = c

c ⊕ "e" = "e" ⊕ key ⊕ "e" = key ⊕ "e" ⊕ "e" = key

If we consider the blank space " " and any text punctuation, the space character has a higher frequency than the character "e". In our test we are going to consider spaces and punctuations, therefore, to break a ciphertext, **you should consider that the most common character is the encryption of the space character.**

Full Example

1- Receive message from server

Message received:

```
003B16A878EAB47EDE31BC3078136F08AE12EFCBD7DDD59EDCCCD1C9D09ED8D1C69ED4CBD3CECD9ED1C8DBCC9
ECAD6DB9ED2DFC4C79EDAD1D99000
```

2- Decode message (containing as data the ciphertext)

message length:	`003B`
md5:	`16A878EAB47EDE31BC3078136F08AE12`
data (ciphertext):	`EFCBD7DDD59EDCCCD1C9D09ED8D1C69ED4CBD3CECD9ED1C8DBCC9ECAD6DB9ED2DFC4C79EDAD1D990`
parity:	`00`

Check if the parity of the message is ok.
Check if the md5 hash of the ciphertext matches the given md5.

3- Break the ciphertext

Find the most common byte in the ciphertext and get the XOR value with the ASCII encoding of space.

most common byte:	0X9E	
space:	0X20	
XOR:	0XBE	(the key is 0xBE)

Decrypt the ciphertext by performing a XOR of each byte from the ciphertext with the key.

Decrypted ASCII-encoded plaintext:
517569636B792062726F776E20666F78206A756D7073206F76657220746865206C617A7920646F672E
Plaintext: **Quick brown fox jumps over the lazy dog.**

4- Encode message (using as data the ASCII-encoded plaintext)

message length:	`003B`
md5:	`BEE544F23ABF58A184FCDF7654BB84DC`
data:	`517569636B792062726F776E20666F78206A756D7073206F76657220746865206C617A7920646F672E`
parity:	`01`

5- Send message to the server

```
003BBEE544F23ABF58A184FCDF7654BB84DC517569636B2062726F776E20666F78206A756D7073206F766572207
46865206C617A7920646F672E01
```

6- Receive confirmation from server

Message received:
`001587B3B64B7DB7419341EF7D493F7709ECF1F501`

7- Decode message with T-Scheme

message length:	`0015`
md5:	`87B3B64B7DB7419341EF7D493F7709EC`
data (ciphertext):	`F1F5`
parity:	`01`

Check if the parity of the message is ok.
Check if the md5 hash of the ciphertext matches the given md5.

8- Decrypt the ciphertext with the previous key

previous key: 0xBE
Decrypted message: 4F4B
Plaintext ASCII encoded: **OK**

Check that you received an **OK** message.

Done!

Questions

If you need any help enrolling for the test or viewing your packets, send an email message to estagiodesenvolvimento@gmail.com.