Técnicas de sustitución y transposición

Kevin Del Castillo Ramírez
Alexander Apaza Torres
Eduardo Basurco Cayhualla

15 de abril de 2019

1. Encriptación alberti

Este algoritmo de encriptación es polialfabético, usando una estructura llamado *disco de Alberti*, que usa dos alfabetos que pueden ser diferentes pero tienen que tener la misma cantidad de carácteres, esta estructura puede ser fácilmente imitada con dos cadenas de misma longitud y simples transposiciones.

Supongamos dos cadenas O e I (*outer disk* e *inner disk* respectivamente), también tenemos una texto plano M, y una clave K, la función se define como

$$E(M_i) = I_{(O_i + K_i \mod l)}$$

donde:

- l es la longitud de ambos discos (i.e. alfabetos).
- M_i, I_i, O_i representan el indice del carácter con respecto a un alfabeto Σ.

Esta definición permite encriptar pequeñas partes del texto original con diferentes "claves" K_i, originalmente podemos rotar el disco inicialmente para encriptar hasta cierta parte, y luego volver a girarlo para encriptar el resto del texto o volver a girar el disco, así hasta encriptar todo el texto plano. Por lo tanto $K = K_1 K_2 K_3 \dots K_n$ donde n es la longitud del texto a encriptar, y cada K_i es un giro de disco, si todos son iguales, se dice que el disco solo gira una vez.

1.1. Código

```
#pragma once

#include <map>
#include <string>
#include <stdexcept>
#include <iostream>
#include <util.hpp>

namespace segcom::alberti {

    using namespace std;
```

```
using segcom::util::modulo;

class AlbertiDisk {
public:
    using InnerSpins = map<size_t, long>;

private:
    wstring inner_disk;
    wstring outer_disk;

public:
    AlbertiDisk() = default;
    AlbertiDisk(wstring const& inner_disk, wstring const& outer_disk) {
        if (inner_disk.length() != outer_disk.length())
            throw runtime_error("Disks are different length");

        this->inner_disk = inner_disk;
        this->outer_disk = outer_disk;
    }

    AlbertiDisk(AlbertiDisk const&) = default;

    wstring encrypt(wstring plain_text, long inner_spin) {
        wstring encrypted_text = L"";

        for (wchar_t chr : plain_text) {
            long outer_index = outer_disk.find(chr);
            if (outer_index == wstring::npos)
                throw runtime_error("Found a character that's not in the disk");

            encrypted_text += inner_disk[modulo(outer_index + inner_spin,
                    inner_disk.length())];
        }

        return encrypted_text;
    }

    wstring variable_encrypt(wstring plain_text, InnerSpins spins) {
        wstring encrypted_text = L"";

        if (spins.rbegin()->first >= plain_text.length())
            throw runtime_error("An index exceeds the length of the plain text");

        if (spins.begin()->first != 0)
            throw runtime_error("Give an initial spin (index = 0)");

        long curr_spin = spins.begin()->second;
        auto next_spin_it = spins.begin(); next_spin_it++;

        for (size_t i = 0; i < plain_text.length();) {
```

```
                long outer_index = outer_disk.find(plain_text[i]);
                if (outer_index == wstring::npos)
                    throw runtime_error("Found a character that's not in the disk");

                encrypted_text += inner_disk[modulo(outer_index + curr_spin,
                        inner_disk.length())];

                i += 1;
                if (next_spin_it != spins.end() && i == next_spin_it->first) {
                    curr_spin += next_spin_it->second;
                    next_spin_it++;
                }
            }

            return encrypted_text;

        }
    };
}

#include <locale>
#include <fstream>
#include <iostream>

#include <prepro.hpp>
#include <alberti.hpp>

using namespace std;
using namespace segcom;

int main(int argc, char** argv) {
    locale::global(locale("en_US.UTF-8")); wcout.imbue(locale());
    wstring alphabet = L"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    if (argc == 1) {
        wcout << "error: no input file!";
        return 1;
    }

    if (argc == 2) {
        wcout << "error: no output file!";
        return 1;
    }

    wifstream inputf(argv[1]);
    wofstream outputf(argv[2]);

    prepro::ReplaceMap replace_map = {
        { L'á', L'a' },
        { L'Á', L'A' },
```

```
        { L'é', L'e' },
        { L'É', L'E' },
        { L'í', L'i' },
        { L'Í', L'I' },
        { L'ó', L'o' },
        { L'Ó', L'O' },
        { L'ú', L'u' },
        { L'Ú', L'U' }
    };

    prepro::EraseVec erase_vec = {
        L' ', L'\t', L'\n', L'¿', L'?',
        L'¡', L'!', L':', L'.', L':', L',',
        L';', L'\"', L'\'', L'"', L'"'
    };

    wstring prepro_text = L"";
    for (wstring line; getline(inputf, line); ) {
        prepro::replace(line, replace_map);
        prepro::to_uppercase(line);
        prepro::erase(line, erase_vec);
        prepro_text += line;
    }

    auto alberti_disk = alberti::AlbertiDisk(alphabet, alphabet);

    long no_spins;
    wcout << "Number of spins: ";
    wcin  >> no_spins;

    long index, step;
    auto inner_spins = alberti::AlbertiDisk::InnerSpins();

    for (long i = 0; i < no_spins; ++i) {
        wcin >> index >> step;
        inner_spins.emplace(index, step);
    }

    auto encrypted_text = alberti_disk.variable_encrypt(prepro_text, inner_spins);
    outputf << encrypted_text;
}
```

1.2. Pruebas

Con el programa anterior probamos encriptar cadenas simples con ambos discos conteniendo el alfabeto $\Sigma = ABCDE \ldots Z$ y giros múltiples de este disco.

1.2.1. Prueba 1

- En la primera prueba el mensaje a encriptar es el siguiente

  > Hola ¿comó estás?, espero que bien. Espero volver a verte pronto.

- La "clave" esta descompuesta por las siguientes reglas

  - Rotación inicial, 5 pasos a la derecha.
  - A partir del quinto carácter, 3 pasos a la izquierda.
  - A partir del décimo carácter, 10 pasos a la derecha.

- El mensaje encriptado es el siguiente

  MTQFHQOQGUBIAMAXMZWYCMJQMVMAXMZWDWTDMZIDMZBMXZWVBW

1.2.2. Prueba 2

- En la segunda prueba el mensaje a encriptar es el siguiente

  > La teoría del Big Bang estipula que el universo se generó por una enorme explosión.

- La "clave" esta descompuesta por las siguientes reglas

  - Rotación inicial, 5 pasos a la derecha.
  - A partir del quinto carácter, 2 pasos a la izquierda.
  - A partir del décimo carácter, 6 pasos a la derecha.
  - A partir del décimo tercero carácter, 3 pasos a la izquierda.

- El mensaje encriptado es el siguiente (agregado salto de linea para poder ser mostrado en este documento)

  QFYJRULDGHUKRMHGTMKYZOVARGWAKKRATOBK XYUYKMKTK-
  XUVUXATGKTUXSKKDVRUYOUT

2. Desencriptación vigenère

El encriptado Vigenère es un algoritmo polialfabético, a menudo se representa como una matriz simétrica donde cada fila representa el alfabeto recorrido un hacia la izquierda en $i-1$ pasos, donde i es el indice de la fila.

La función de desencriptación teniendo en cuenta un alfabeto Σ de longitud l, y una clave con longitud m, con C siendo el mensaje encriptado y K la llave, se define como

$$D_K(C_i) = (C_i - K_{(i \mod m)}) \mod l$$

donde C_i, K_i representan el indice del carácter con respecto al alfabeto Σ.

	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

Figura 1: Tabula recta

2.1. Código

```
#pragma  once

#include <iostream>
#include <string>
#include <stdexcept>

#include <util.hpp>

namespace segcom::vignere {
    using namespace std;
    using segcom::util::modulo;

    class VigenereCipher {
    private:
        wstring alphabet;

    public:
        VigenereCipher() = default;
        VigenereCipher(wstring const& alphabet): alphabet(alphabet) {}
        VigenereCipher(VigenereCipher const&) = default;

        wstring decrypt(wstring const& encrypted_text, wstring key) {
            wstring plain_text = L"";

            for (auto chr: key)
                if (alphabet.find(chr) == wstring::npos)
                    throw runtime_error("key contains characters not included in the\
                            alphabet");
```

```
            for (auto chr: encrypted_text)
                if (alphabet.find(chr) == wstring::npos)
                    throw runtime_error("encrypted text contains characters not included\
                            in the alphabet");

            for (auto i = 0; i < encrypted_text.length(); ++i) {
                long encrypted_index = alphabet.find(encrypted_text[i]);
                long key_index = alphabet.find(key[modulo(i, key.length())]);

                plain_text += alphabet[modulo(encrypted_index - key_index,
                        alphabet.length())];
            }

            return plain_text;
        }
    };
}

#include <locale>
#include <fstream>
#include <iostream>

#include <prepro.hpp>
#include <vigenere.hpp>

using namespace std;
using namespace segcom;

int main(int argc, char** argv) {
    locale::global(locale("en_US.UTF-8")); wcout.imbue(locale());
    wstring alphabet = L"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    if (argc == 1) {
        wcout << "error: no input file!";
        return 1;
    }

    if (argc == 2) {
        wcout << "error: no output file!";
        return 1;
    }

    wifstream inputf(argv[1]);
    wofstream outputf(argv[2]);

    wstring encrypted_text;
    getline(inputf, encrypted_text);

    wstring key;
```

```
    wcout << "Key: ";
    wcin  >> key;

    auto vigenere_cipher = vignere::VigenereCipher(alphabet);
    auto plain_text = vigenere_cipher.decrypt(encrypted_text, key);

    outputf << plain_text;
}
```

2.2. Pruebas

Con el programa anterior probamos desencriptar cadenas simples con diferentes claves

2.2.1. Prueba 1

- En la primera prueba el mensaje a desencriptar es el siguiente

  WEEUELRMRVWPMGFZRNIRYEEMQPPUQVZWMG

- La clave es “LEMON”
- El mensaje desencriptado es el siguiente

  LASGRANADILLASSONBUENASYDELICIOSAS

2.2.2. Prueba 2

- En la segunda prueba el mensaje a desencriptar es el siguiente

  PPMHNQXIDIDSNLLDLVMOYAWECHQQRR

- La clave es “LEMONADE”
- El mensaje desencriptado es el siguiente

  ELATAQUESEREALIZARAALATARDECER