

Aluno: Guilherme Quadros da Silva

RU: 3282910

Uninter - Matemática Computacional: AP Criptografia Simétrica com XOR

Atividade Prática (AP) da disciplina de Matemática Computacional do curso de Engenharia de Computação da Uninter:

Codificar a mensagem "APROVADO" por criptografia simétrica pelo algoritmo elementar XOR utilizando como chave criptográfica o seu RU ou parte dele. Após a obtenção da cifra decodificá-la comprovando a reciprocidade do processo.

Instruções

A pasta "**source**" contém o código-fonte do programa criado para resolver o problema proposto. A solução do projeto foi escrita em .NET 4.6 com a linguagem de programação C# utilizando a IDE "Rider" versão 2020.1.3 da JetBrains, mas a última versão do Visual Studio Community deve conseguir abrir o projeto normalmente. Caso queira somente visualizar o código-fonte basta abrir o arquivo "Program.cs" em um bloco de notas.

A pasta "**exe**" contém um executável do programa desenvolvido compatível .NET 4.6, o Windows 10 deve suportar essa versão por padrão: <https://docs.microsoft.com/en-us/archive/blogs/astebner/mailbag-what-version-of-the-net-framework-is-included-in-what-version-of-the-os>

Mas se o seu sistema operacional não suportar o executável você conseguirá facilmente achar na internet uma versão de .NET Framework ou Mono compatível.

O código é escrito em inglês por uma preferência minha e costume mesmo.

O Programa

Se você está lendo somente o PDF desse trabalho, o código, o executável e todo o resto também podem ser encontrados no seguinte repositório:

<https://github.com/guiquadros/uninter-xor-crypto>

Abaixo está a classe "Program.cs" contendo todo o código-fonte do programa. Este projeto contém dependência com a biblioteca "Newtonsoft.Json", você consegue encontrá-la facilmente na internet ou instalá-la via NuGet package manager na sua IDE de preferência, a DLL dela já está inclusa projeto que está no repositório acima mencionado.

```
// #define DEBUG_BIN_CONVERSION

using System;
using System.Collections.Generic;
using System.Threading;
using Newtonsoft.Json;

namespace xor_cryptography
{
    public class Program
    {
        private const int BIN_PART_SIZE = 7;

        public static void Main(string[] args)
        {
            const string TEXT_TO_ENCRYPT = "APROVADO";
            const int RU = 3282910;

            Console.ForegroundColor = ConsoleColor.Yellow;

            Console.WriteLine("Uninter - Matematica Computacional: AP Criptografia Simetrica com XOR");
            Console.WriteLine("Autor: Guilherme Quadros da Silva");

            // encryption
            WaitNextStep("PRESSIONE UMA TECLA PARA INICIAR O PROGRAMA DE CRIPTOGRAFIA E DESCRIPTOGRAFIA.");
            WriteLineInDifferentColor($"Encriptando \"{TEXT_TO_ENCRYPT}\" com o RU \"{RU}\"...", ConsoleColor.Red);
            Console.WriteLine();
            string encryptedString = EncryptStringXor(TEXT_TO_ENCRYPT, RU);
            Console.WriteLine();
            Console.WriteLine();
            WriteInDifferentColor($"\"{TEXT_TO_ENCRYPT}\" ", ConsoleColor.Cyan);
            WriteInDifferentColor($"foi criptografado para: ", ConsoleColor.Red);
            WriteLineInDifferentColor($"{ToLiteral(encryptedString)}", ConsoleColor.Cyan);
            Console.WriteLine();

            // decryption
            WaitNextStep("PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE
```

```

DESCRIPTOGRAFIA.");
    WriteLineInDifferentColor($"Descriptando {ToLiteral(encryptedString)} com
o RU \"{RU}\"...", ConsoleColor.Red);
    string decryptedString = EncryptStringXor(encryptedString, RU);
    Console.WriteLine();
    Console.WriteLine();
    WriteInDifferentColor($" {ToLiteral(encryptedString)} ",
ConsoleColor.Cyan);
    WriteInDifferentColor($"foi descriptografado para: ", ConsoleColor.Red);
    WriteLineInDifferentColor($"\"{decryptedString}\"", ConsoleColor.Cyan);

    Console.WriteLine();
    WaitNextStep("PRESSIONE QUALQUER TECLA PARA FECHAR A EXECUCAO DO
PROGRAMA.");
}

private static string EncryptStringXor(string textToEncrypt, int RU)
{
    if (string.IsNullOrEmpty(textToEncrypt)) return string.Empty;

    Console.WriteLine($"Buscando valores de {ToLiteral(textToEncrypt)} na
tabela ASCII:");
    Console.WriteLine($"\"{character}\" = \"{valor ASCII em decimal}\" e \"{
valor ASCII em binario}\"");
    List<string> binCharsListStr = new List<string>();
    foreach (char character in textToEncrypt)
    {
        int charAsDec = character;
        string charAsBinStr = ConvertDecToBinStr(charAsDec);

        string decToShow = $"\"{charAsDec} (10)\"";
        WriteLineInDifferentColor($"
{ToLiteral(character.ToString()).PadLeft(10, ' ')} = {decToShow.PadLeft(10, ' ')} e
\"{charAsBinStr} (2)\"", ConsoleColor.Gray);

        binCharsListStr.Add(charAsBinStr);
    }

    string binTxtStr = string.Join(string.Empty, binCharsListStr);
    Console.WriteLine();
    WriteLineInDifferentColor($" {ToLiteral(textToEncrypt)} = \"{binTxtStr}
(2)\"", ConsoleColor.Red);

    WaitNextStep("PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE OBTENCAO DA
CHAVE DE CRIPTOGRAFIA A PARTIR DO RU.");

    ulong keyInDec = GetCriptoKeyFromRU(RU, binTxtStr, textToEncrypt);

```

```

    string keyInBinStr = ConvertDecToBinStr(keyInDec);
    Console.WriteLine();
    WriteLineInDifferentColor($"Chave obtida = \"{keyInBinStr} (2)\",
ConsoleColor.Red);
    Console.WriteLine();

    int diffKeyAndTxt = keyInBinStr.Length - binTxtStr.Length;

    // adding additional "0" in the key
    if (diffKeyAndTxt > 0)
    {
        int diffWithPartSize = BIN_PART_SIZE - diffKeyAndTxt;

        if (diffWithPartSize > 0)
        {
            diffKeyAndTxt += diffWithPartSize;

            // complete key with "0" on the left
            keyInBinStr = keyInBinStr.PadLeft(keyInBinStr.Length +
diffWithPartSize, '0');
        }
    }

    // encryption logic started
    WaitNextStep("PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE CIFRAGEM.");
    Console.WriteLine("Criptografando com XOR:");
    const string XOR_SEPARATOR = " ";

    for (int i = 0; i < diffKeyAndTxt; i++)
    {
        Console.Write(" ");
    }
    Console.Write(XOR_SEPARATOR);
    foreach (string binChar in binCharsListStr)
    {
        WriteInDifferentColor(XOR_SEPARATOR + binChar, ConsoleColor.Gray);
    }
    WriteInDifferentColor($"{{XOR_SEPARATOR}}{{ToLiteral(textToEncrypt}})",
ConsoleColor.Cyan);

    Console.WriteLine();
    WriteInDifferentColor("XOR ", ConsoleColor.Cyan);
    for (var i = 0; i < binCharsListStr.Count; i++)
    {
        string keyPart = keyInBinStr.Substring((i *
binCharsListStr[i].Length) + diffKeyAndTxt, binCharsListStr[i].Length) +
XOR_SEPARATOR;

```

```

        if (i == 0 && diffKeyAndTxt > 0)
        {
            string exceededKeyPart = keyInBinStr.Substring(0, diffKeyAndTxt);
            WriteInDifferentColor(exceededKeyPart, ConsoleColor.DarkBlue);
            Console.Write(XOR_SEPARATOR);
        }

        WriteInDifferentColor(keyPart, ConsoleColor.Gray);
    }
    WriteInDifferentColor($"(\"{keyInDec}\")", ConsoleColor.Cyan);

    Console.WriteLine();
    Console.Write(XOR_SEPARATOR);
    for (var i = 0; i < binTxtStr.Length + (binCharsListStr.Count *
XOR_SEPARATOR.Length) + diffKeyAndTxt + XOR_SEPARATOR.Length; i++)
    {
        Console.Write("-");
    }

    Console.WriteLine();
    Console.Write(XOR_SEPARATOR);
    if (diffKeyAndTxt > 0)
    {
        for (int i = 0; i < diffKeyAndTxt; i++)
        {
            Console.Write(" ");
        }

        Console.Write(XOR_SEPARATOR);
    }

    // the actual encryption
    string encryptedBinTxt = string.Empty;
    for (int i = binTxtStr.Length - 1; i >= 0; i--)
    {
        int txtCharAsDec = int.Parse(binTxtStr[i].ToString());
        int keyCharAsDec = int.Parse(keyInBinStr[i +
diffKeyAndTxt].ToString());

        string xorResultStr = (txtCharAsDec ^ keyCharAsDec).ToString();
        encryptedBinTxt = $"{xorResultStr}{encryptedBinTxt}";
    }

    string encryptedTxt = string.Empty;
    string characterInBin = string.Empty;
    // write down the result

```

```

        for (int i = 0; i < encryptedBinTxt.Length; i++)
        {
            characterInBin = $"{characterInBin}{encryptedBinTxt[i]}";

            if ((i + 1) % BIN_PART_SIZE == 0)
            {
                WriteInDifferentColor($"{characterInBin}", ConsoleColor.Red);

                int characterInDec = ConvertBinToDec(characterInBin);
                char character = (char) characterInDec;
                encryptedTxt = $"{encryptedTxt}{character}";

                Console.Write(XOR_SEPARATOR);
                characterInBin = string.Empty;
            }
        }
        WriteInDifferentColor($"({ToLiteral(encryptedTxt)})", ConsoleColor.Cyan);
        return encryptedTxt;
    }

    private static void WaitNextStep(string message)
    {
        Console.WriteLine();
        WriteLineInDifferentColor(message, ConsoleColor.Magenta);
        Console.ReadKey(true);
        Console.WriteLine();
    }

    /// <summary>
    /// Creates the cryptography key with a useful size (compared to the string
    that will be encrypted) based on RU.
    /// </summary>
    /// <param name="RU">Student RU.</param>
    /// <param name="binStr"></param>
    /// <param name="textToEncrypt"></param>
    /// <param name="binCharsListStr"></param>
    /// <returns>The cryptography key.</returns>
    private static ulong GetCriptoKeyFromRU(int RU, string binStr, string
textToEncrypt)
    {
        Console.WriteLine($"Obtendo a chave de criptografia pelo RU \"{RU}\":"");

        ulong keyInDec = (ulong)RU;
        string RUstr = RU.ToString();
        string RUinBinStr = ConvertDecToBinStr(RU);
        string keyInBinStr = RUinBinStr;
        Console.WriteLine($"\"{keyInDec} (10)\n = \"{keyInBinStr} (2)\n");
    }

```

```

int tryCount = 0;
int RUcharPos = -1;

// Repeats this until we find a proper key
do
{
    // waits a little to display each info (too much info to display at
once)
    Thread.Sleep(300);
    WriteInDifferentColor($"{++tryCount}) Comparando o tamanho da chave
\"{keyInDec} (10)\" (\\"{keyInBinStr} (2)\" ", ConsoleColor.Cyan);
    WriteInDifferentColor($"[{keyInBinStr.Length} bits]",
ConsoleColor.Yellow);
    WriteInDifferentColor($" obtida com a string que vai ser
criptografada {ToLiteral(textToEncrypt)} (\\"{binStr} (2)\" ", ConsoleColor.Cyan);
    WriteInDifferentColor($"[{binStr.Length} bits]",
ConsoleColor.Yellow);
    WriteLineInDifferentColor($".", ConsoleColor.Cyan);

    Console.Write($"    {keyInBinStr.Length} >= {binStr.Length}? --> ");
    if (keyInBinStr.Length >= binStr.Length)
    {
        WriteLineInDifferentColor("Sim! ", ConsoleColor.Red);
        break;
    }

    WriteLineInDifferentColor($"Nao! O tamanho da chave obtida nao e
suficiente para criptografar com XOR a string {ToLiteral(textToEncrypt)}.",
ConsoleColor.Blue);
    // concatenates the current key with the RU in the end to generate a
bigger keys
    char nextRUDigit = GetNextDigitFromRU(RUstr, ref RUcharPos);
    keyInDec = ulong.Parse($"{keyInDec}{nextRUDigit}");
    Console.Write($"    Concatenando a chave com o proximo digito do RU
(");
    WriteInDifferentColor($"\"{keyInDec} (10)\" + \"{nextRUDigit}
(10)\""", ConsoleColor.Gray);
    Console.WriteLine($" para obter uma chave maior:");

    keyInBinStr = ConvertDecToBinStr(keyInDec);
    WriteLineInDifferentColor($"    \"{keyInDec} (10)\" = \"{keyInBinStr}
(2)\""", ConsoleColor.Gray);
} while (true);

return keyInDec;
}

```

```

private static void WriteLineInDifferentColor(string message, ConsoleColor
color)
{
    ConsoleColor oldFgColor = Console.ForegroundColor;
    Console.ForegroundColor = color;
    Console.WriteLine(message);
    Console.ForegroundColor = oldFgColor;
}

private static void WriteInDifferentColor(string message, ConsoleColor color)
{
    ConsoleColor oldFgColor = Console.ForegroundColor;
    Console.ForegroundColor = color;
    Console.Write(message);
    Console.ForegroundColor = oldFgColor;
}

/// <summary>
/// Return the next digit position in the student's RU.
/// </summary>
/// <param name="RUstr">The student's RU.</param>
/// <param name="digitPos">The digit used in last concatenation.</param>
/// <returns></returns>
private static char GetNextDigitFromRU(string RUstr, ref int digitPos)
{
    if (digitPos < RUstr.Length - 1)
    {
        digitPos++;
    }
    else
    {
        digitPos = 0;
    }

    return RUstr[digitPos];
}

private static string ConvertDecToBinStr(int decNum)
{
    return ConvertDecToBinStr((ulong) decNum);
}

private static string ConvertDecToBinStr(ulong decNum)
{
    #if DEBUG_BIN_CONVERSION
        Console.WriteLine($"Convertendo \"{decNum}\" para binario:");
    #endif
}

```



```

#endif

        ulong result = decNum;
        string binResult = string.Empty;

        while (result > 1)
        {
            #if DEBUG_BIN_CONVERSION
                Console.Write($"{result} dividido por 2: ");
            #endif

            ulong remaining = result % 2;
            binResult = $"{remaining}{binResult}";
            result /= 2;

            #if DEBUG_BIN_CONVERSION
                Console.WriteLine($"quociente = {result}; resto = {remaining}");
            #endif
        }

        binResult = $"{result}{binResult}";

        binResult = binResult.PadLeft(BIN_PART_SIZE, '0');

        #if DEBUG_BIN_CONVERSION
            Console.WriteLine($"\"{decNum}\" em binario = \"{binResult}\"");
        #endif

        return binResult;
    }

    private static int ConvertBinToDec(string binNum)
    {
        const int NEW_BASE = 2;

        // only works with integers (without the floating point piece)
        int lastIndex = binNum.Length - 1;
        int result = 0;
        for (int i = lastIndex; i >= 0; i--)
        {
            int index = lastIndex - i;
            int digit = int.Parse(binNum[index].ToString());
            result += digit * (int)Math.Pow(NEW_BASE, i);
        }

        return result;
    }
}

```

```

    /// <summary>
    /// Helps showing the special characters on the screen./
    /// </summary>
    private static string ToLiteral(string input)
    {
        return JsonConvert.SerializeObject(input);
    }
}
}

```

Para executar o programa siga os passos abaixo.

1. O programa pode ser iniciado rodando o arquivo "xor_cryptography.exe" dentro da pasta "exe". Cada etapa do processo ele espera um comando do usuário para prosseguir.



```

guiquadros@iMac848 uninter-xor-crypto (master) $ ls -la
total 40
drwxr-xr-x  8 guiquadros  staff   256 Aug 31 04:34 .
drwxrwxrwx 29 guiquadros  staff   928 Aug 29 23:59 ..
-rw-r--r--@ 1 guiquadros  staff  6148 Aug 31 04:33 .DS_Store
drwxr-xr-x 16 guiquadros  staff   512 Aug 31 04:34 .git
-rw-r--r--  1 guiquadros  staff  6042 Aug 29 23:59 .gitignore
-rw-r--r--  1 guiquadros  staff  1456 Aug 31 04:34 README.md
drwxr-xr-x  5 guiquadros  staff   160 Aug 31 04:30 exe
drwxr-xr-x  4 guiquadros  staff   128 Aug 30 00:00 source
guiquadros@iMac848 uninter-xor-crypto (master) $ mono exe/xor_cryptography.exe
Uninter - Matematica Computacional: AP Criptografia Simetrica com XOR
Autor: Guilherme Quadros da Silva

PRESSIONE UMA TECLA PARA INICIAR O PROGRAMA DE CRIPTOGRAFIA E DESCRIPTOGRAFIA.

```

2. A primeira parte é a criptografia, que é iniciada percorrendo cada caracter da palavra "APROVADO" e obtendo seu valor na tabela ASCII em decimal e depois convertendo cada valor decimal para o seu correspondente em binário:

no passo anterior tem 59 bits enquanto a palavra "APROVADO" em binário tem 56 bits na conversão utilizada no passo 2).

```

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE CIFRAGEM.

Criptografando com XOR:
      1000001  1010000  1010010  1001111  1010110  1000001  1000100  1001111  ("APROVADO")
XOR 0000100  1000111  0010100  1011110  1000111  1010101  1100110  0011001  ("328291032829103282")
-----
      0000110  1000100  0001100  0001000  0000011  0100111  1011101  1111101  ("\\u0006D\\f\\b\\u0003'j}")

"APROVADO" foi criptografado para: "\\u0006D\\f\\b\\u0003'j}"

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE DESCRIPTOGRAFIA.

```

5. A descriptografia segue os mesmos passos da criptografia, ela só passa a string cifrada ao invés de "APROVADO" para a mesma rotina, utilizando o mesmo RU ("3282910") como base para obtenção da chave. Abaixo a conversão de cada caracter cifrado para binário:

```

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE DESCRIPTOGRAFIA.

Descriptando "\\u0006D\\f\\b\\u0003'j}" com o RU "3282910"...
Buscando valores de "\\u0006D\\f\\b\\u0003'j}" na tabela ASCII:
"{character}" = "{valor ASCII em decimal}" e "{valor ASCII em binario}"
  "\\u0006" = "6 (10)" e "0000110 (2)"
    "D" = "68 (10)" e "1000100 (2)"
    "\\f" = "12 (10)" e "0001100 (2)"
    "\\b" = "8 (10)" e "0001000 (2)"
  "\\u0003" = "3 (10)" e "0000011 (2)"
    "'" = "39 (10)" e "0100111 (2)"
    "j" = "93 (10)" e "1011101 (2)"
    "}" = "125 (10)" e "1111101 (2)"

  "\\u0006D\\f\\b\\u0003'j}" = "0000110100010000011000001000000001101001111011111101 (2)"

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE OBTENCAO DA CHAVE DE CRIPTOGRAFIA A PARTIR DO RU.

```

6. Obtenção da chave a partir do número do RU seguindo a mesma lógica anterior:

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE OBTENÇÃO DA CHAVE DE CRIPTOGRAFIA A PARTIR DO RU.

[illegible]

```
Chave obtida = "10010001110010100101111010001111010101110011000110010110010 (2)"
```

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE CIFRAGEM.

1

7. O resultado final usando o operador XOR, provando que é possível obter a string "APROVADO" novamente a partir da string cifrada.

PRESSIONE UMA TECLA PARA INICIAR O PROCESSO DE CIFRAGEM.

Criptografando com XOR:

	0000110	1000100	0001100	0001000	0000011	0100111	1011101	1111101	("v0000Dxfv\0003"]")
XOR 0000100	1000111	0010100	1011110	1000111	1010101	1100110	0011001	0110010	("328291032829103282")
	1000001	1010000	1010010	1001111	1010110	1000001	1000100	1001111	("APROVADO")

"\u0006D\f\b\u0003'j}" foi descryptografado para: "APROVADO"

PRESSIONE QUALQUER TECLA PARA FECHAR A EXECUCAO DO PROGRAMA.