UNIVERSIDADE NOVE DE JULHO - UNINOVE
NOME DO PROJETO EM COMPUTAÇÃO APLICADA

EXCLUÍDOS OS DADOS SOBRE OS AUTORES EM ATENDIMENTO A LGPD - LEI GERAL DE PROTEÇÃO DE DADOS

CUBO DE RUBIK

São Paulo
2024

EXCLUÍDOS OS DADOS SOBRE OS AUTORES EM ATENDIMENTO A LGPD - LEI GERAL DE PROTEÇÃO DE DADOS

CUBO DE RUBIK

Projeto apresentado a Universidade Nove de Julho - UNINOVE, como parte dos requisitos obrigatórios para obtenção do título de Bacharel em Ciência da Computação.

Prof. Orientador: Edson Melo de Souza, Dr.

São Paulo
2024

RESUMO

Este documento descreve uma aplicação de criptografia inovadora que utiliza um sistema de matrizes inspirado no famoso quebra-cabeça tridimensional, o Cubo de Rubik. A abordagem de criptografia aproveita a complexidade única do cubo, transformando-o em um método eficaz de encriptação. A chave criptográfica essencial é derivada do horário do sistema da máquina do usuário, adicionando uma camada adicional de segurança. A implementação em C# proporciona uma base sólida e versátil para o desenvolvimento do projeto, garantindo compatibilidade e desempenho. Essa combinação de técnicas resulta em uma solução robusta e inovadora para a segurança da informação.

ABSTRACT

This documentation describes a new cryptography application that uses the matrix system bases on the famous puzzle three-dimensional, the Rubik Cube. The approach of the cryptography takes the unique complexity of the cube, transforming it into a effective method of encryption. The essential cryptography key come from the time of the user's machine, adding a extra layer of security. The implantation of C# gives a solid and versatile base to the development of the project, ensuring compatibility and performance. This combination of techniques results on a sturdy and innovative solution to the information security.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

1 INTRODUÇÃO

O design da aplicação foi concebido e desenvolvido pela presente equipe, que empregou um modelo de mapeamento exclusivo, desenhado com a assistência de um Cubo de Rubik físico para uma autenticidade e precisão meticulosas. No coração desta aplicação, reside a representação de cada segmento do cubo, categorizado de acordo com as seis cores distintas presentes em suas respectivas faces. Apesar de um movimento, ou "giro", do cubo resultar na alteração da posição das peças, as identidades (cores) desses elementos permanecem inalteradas. O processo de criptografia é impulsionado pela chave do usuário, composta por seis componentes temporais: ano, mês, dia, hora, minuto e segundo. A inserção desses dados pelo usuário inicia uma sequência de seis giros meticulosos, correspondentes a cada unidade de tempo, aplicados às seis faces do cubo simulado. Esta operação é a essência da criptografia: cada "giro"representa uma permutação dos valores textuais, uma dança coordenada de caracteres que trocam de posições de maneira controlada e complexa. Para viabilizar esta abordagem, estabelece-se um arranjo matricial que captura e cataloga os valores de cada elemento do cubo, armazenando cinquenta e quatro unidades de dados em seis matrizes distintas. Essas matrizes servem como um registro organizado e acessível dos dados processados. Com a solicitação de criptografia pelo usuário, o sistema ativa o mecanismo de "giros"do cubo, baseando-se nas chaves definidas para transposição dos dados textuais entre as matrizes. A engenhosidade da aplicação não se limita à criptografia. A descriptografia é também uma função integral deste sistema, revertendo o texto ao seu estado original. Tal processo é alcançado ao executar os "giros"em sentido inverso, desembaralhando a sequência de movimentos previamente aplicados e restaurando a ordem inicial dos caracteres. A habilidade do algoritmo de reverter com precisão cada ação criptográfica é um testemunho da sua sofisticação e robustez, garantindo a integridade dos dados e a confidencialidade das informações do usuário. A documentação a seguir oferece uma visão detalhada sobre a implementação técnica, a lógica subjacente e as instruções operacionais deste sistema de criptografia, refletindo nosso compromisso com a excelência em segurança de dados e inovação em criptografia.

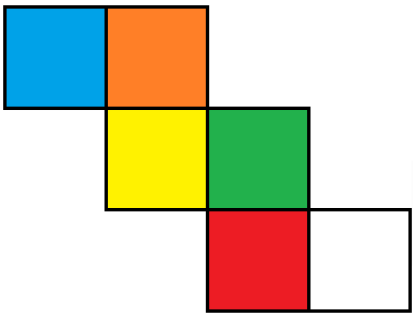


Figura 1.1 – *Cubo plano*

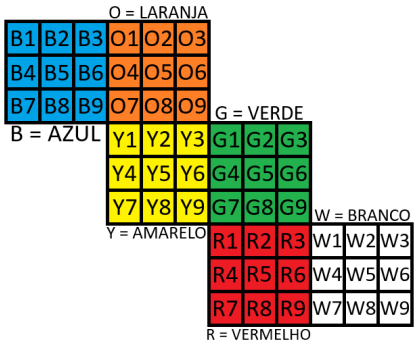


Figura 1.2 – *Cubo mapeado*

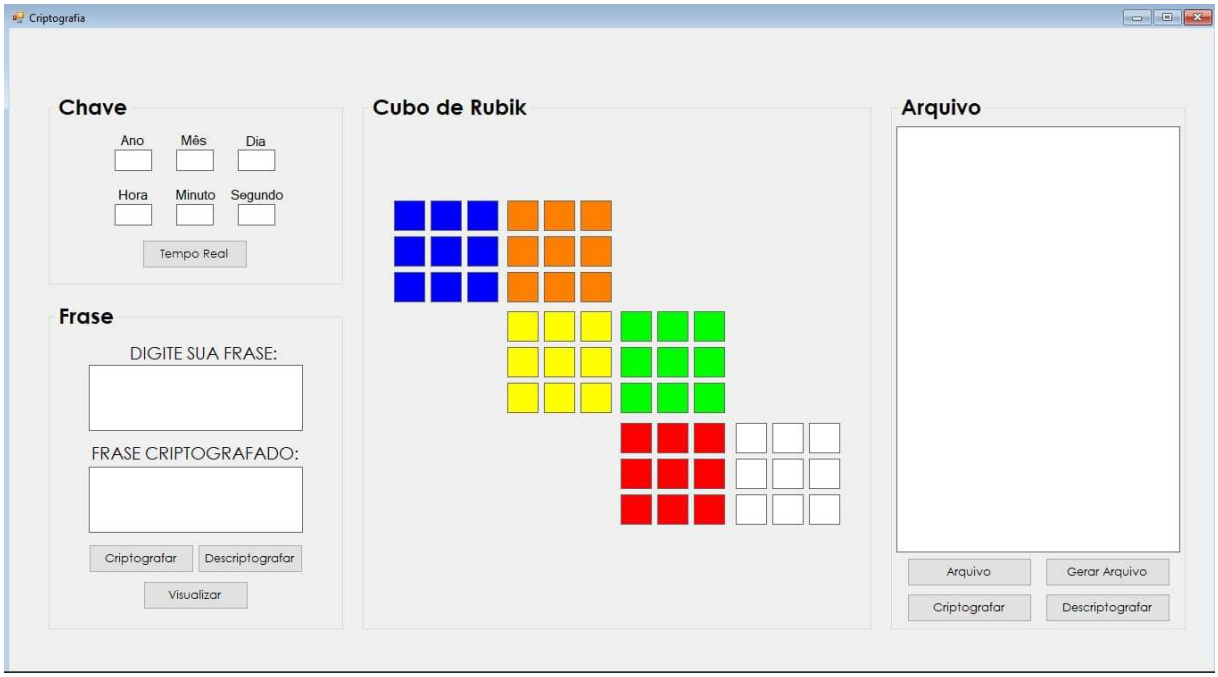
Figura 1.3 – *Mapeamento do cubo em forma de tabela*

B1	->	B3	O1	->	O3	Y1	->	Y3	G1	->	G3	R1	->	R3	W1	->	W3
B2	->	B6	O2	->	O6	Y2	->	Y6	G2	->	G6	R2	->	R6	W2	->	W6
B3	->	B9	O3	->	O9	Y3	->	Y9	G3	->	G9	R3	->	R9	W3	->	W9
B4	->	B2	O4	->	O2	Y4	->	Y2	G4	->	G2	R4	->	R2	W4	->	W2
B5	->	B5	O5	->	O5	Y5	->	Y5	G5	->	G5	R5	->	R5	W5	->	W5
B6	->	B8	O6	->	O8	Y6	->	Y8	G6	->	G8	R6	->	R8	W6	->	W8
B7	->	B1	O7	->	O1	Y7	->	Y1	G7	->	G1	R7	->	R1	W7	->	W1
B8	->	B4	O8	->	O4	Y8	->	Y4	G8	->	G4	R8	->	R4	W8	->	W4
B9	->	B7	O9	->	O7	Y9	->	Y7	G9	->	G7	R9	->	R7	W9	->	W7
R7	->	W7	B3	->	W3	B7	->	O7	Y3	->	O3	Y7	->	G7	R3	->	G3
R8	->	W8	B6	->	W6	B8	->	O8	Y6	->	O6	Y8	->	G8	R6	->	G6
R9	->	W9	B9	->	W9	B9	->	O9	Y9	->	O9	Y9	->	G9	R9	->	G9
W7	->	O1	W3	->	G1	O7	->	G1	O3	->	W1	G7	->	W1	G3	->	O1
W8	->	O4	W6	->	G2	O8	->	G4	O6	->	W2	G8	->	W4	G6	->	O2
W9	->	O7	W9	->	G3	O9	->	G7	O9	->	W3	G9	->	W7	G9	->	O3
O1	->	Y1	G1	->	Y1	G1	->	R1	W1	->	R1	W1	->	B1	O1	->	B1
O4	->	Y4	G2	->	Y2	G4	->	R4	W2	->	R2	W4	->	B4	O2	->	B2
O7	->	Y7	G3	->	Y3	G7	->	R7	W3	->	R3	W7	->	B7	O3	->	B3
Y1	->	R7	Y1	->	B3	R1	->	B7	R1	->	Y3	B1	->	Y7	B1	->	R3
Y4	->	R8	Y2	->	B6	R4	->	B8	R2	->	Y6	B4	->	Y8	B2	->	R6
Y7	->	R9	Y3	->	B9	R7	->	B9	R3	->	Y9	B7	->	Y9	B3	->	R9

Fonte: Autor

2 INTERFACE DA APLICAÇÃO

Figura 2.1 – *Interface*



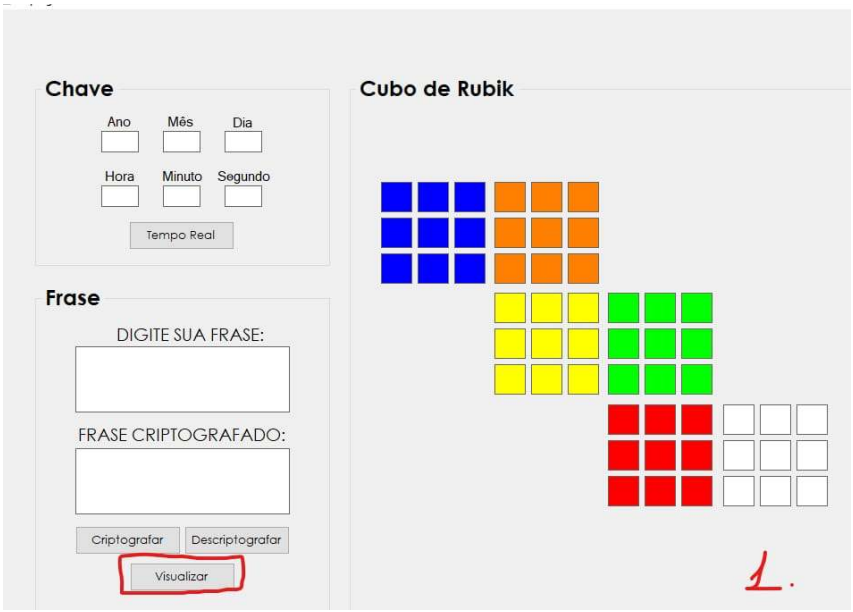
Fonte: Autor

2.1 Interface

A interface da aplicação é intuitiva e objetiva, o que facilita a navegação e a operação pelo usuário. A tela principal é claramente dividida em três seções principais, cada uma com uma função distinta, refletindo uma abordagem modular ao design da interface. Abaixo detalha-se cada área e suas funcionalidades:

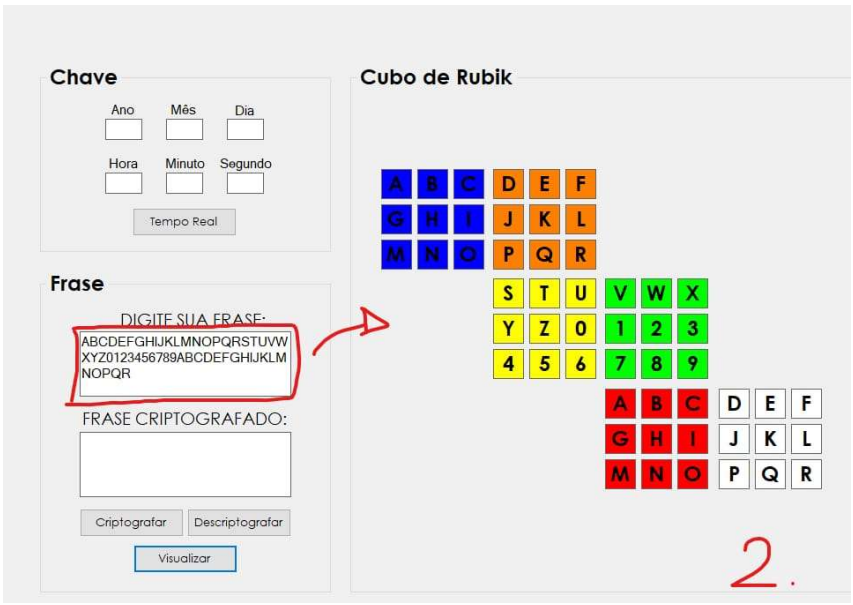
3 BOTÃO VISUALIZAR

Figura 3.1 – *Botão visualizar*



Fonte: Autor

Figura 3.2 – *Frase posta no cubo*



Fonte: Autor

3.0.1 Explicação do Botão

Este botão trata-se do mecanismo pelo qual a aplicação prepara o texto para ser visualizado e potencialmente criptografado, garantindo que ele tenha o tamanho correto para ser processado pelo algoritmo. A interface visual associada mostra como o texto é mapeado para o Cubo de Rubik, proporcionando ao usuário uma representação gráfica das operações de criptografia. A aplicação se beneficia de validar a entrada antes de proceder com a criptografia, evitando erros e garantindo que o texto seja apropriadamente manipulado. Esta atenção aos detalhes é fundamental em uma aplicação de segurança, onde a precisão da entrada afeta diretamente a saída do processo de criptografia. Para uma compreensão concisa, cada artifício será separado por tópicos e subtópicos:

3.0.2 Declaração de Matrizes

- O sistema começa com a declaração de várias matrizes de strings, cada uma representando um lado do Cubo de Rubik.
- No contexto do código, string[,] define uma matriz de strings com duas dimensões, e blue é uma instância dessa matriz, tendo 3 linhas e 3 colunas, perfazendo um total de 9 strings para armazenamento.
- Cada matriz é inicializada para conter 9 posições (new string[3, 3]), e cada posição é destinada a armazenar um caractere ou conjunto de caracteres, possivelmente representando as cores de uma peça do Cubo de Rubik.

3.0.3 String 'caesar'

- Uma string especial chamada caesar contém a sequência de caracteres que serão alocados nas matrizes declaradas. Esta string é uma referência à Cifra de César, uma técnica clássica de criptografia que desloca letras ao longo do alfabeto.

3.0.4 Condicional para Verificação e Preenchimento

- Uma condição verifica se o comprimento do texto fornecido pelo usuário é menor que 54 caracteres.
- Se for menor, o código calcula quantos caracteres adicionais são necessários (caracteresRestantes) e concatena esses caracteres ao Texto.Text. A cadeia usada para o preenchimento vem da string caesar e é recortada para o número exato de caracteres necessários para alcançar o total de 54.

```
1     if (Texto.Text.Length < 54) {
```

```
2           int caracteresRestantes = 54 - Texto.Text.Length;
3           Texto.Text += "ABCDEFGHIJKLMNOPQRSTUVWXYZ
4           0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789A
5           BCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".
6           Substring(0, caracteresRestantes);
7       }
```

3.0.5 Exceção para Textos Longos

- Se o texto exceder 54 caracteres, uma mensagem de erro é exibida, informando ao usuário que a mensagem é longa demais para ser criptografada. O código então executa a função LimparTextBoxes(), que limpa os campos de texto, e sai da função para impedir a execução de mais código.
- Adiciona um loop onde caso o total de caracteres ser maior que 54, uma mensagem de erro será exibida para o usuário.
- Após isso o void LimparTextBoxes() é acionado e caixa de texto é limpa.

```
1       else {
2           while (Texto.Text.Length > 54) {
3               MessageBox.Show("Sua Mensagem longa demais para criptografar
                    .",
4               "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
5               LimparTextBoxes();
6               Texto.Text = "";
7           return;
8           }
9       }
```

3.1 Método Aloca

```
1       void Aloca(string Texto, string[,] fcolor, string[,] lcolor) {
2           for (int J = 0; J < 3; J++) {
3               for (int I = 0; I < 6; I++) {
4                   if (I < 3) {
5                   fcolor[J, I] = Texto.Substring(I + (J * 6), 1);
6               } else {
7                   lcolor[J, I - 3] = Texto.Substring(I + (J * 6), 1);
8               }
9               }
10          }
11      }
```

3.1.1 Estrutura do Método

- O método Aloca é definido para receber três argumentos: uma string Texto e duas matrizes bidimensionais fcolor e lcolor, que representam, respectivamente, as faces frontal e lateral do cubo no contexto da criptografia.
- fcolor é preenchida com os caracteres da primeira metade de cada conjunto de seis caracteres, enquanto lcolor recebe a segunda metade.

3.1.2 Lógica de Alocação

- A função começa iterando sobre as linhas (J) e colunas (I) das matrizes. A iteração ocorre primeiramente através de cada linha (J) e depois por cada um dos seis caracteres (I) nos grupos de seis da string Texto.
- Para as colunas de 0 a 2 (I $<$ 3), os caracteres são alocados na matriz fcolor. Para as colunas de 3 a 5, os caracteres são alocados na matriz lcolor, ajustando o índice da coluna com I - 3.
- Com a lógica aplicada, converte-se uma matriz bidimensional em unidimensional.

3.1.3 Função na Criptografia

- O preenchimento das matrizes é equivalente a um "giro"do cubo na criptografia. Cada matriz fcolor e lcolor recebe um conjunto específico de caracteres, assim como cada face de um Cubo de Rubik teria um conjunto específico de cores.
- Este método não apenas prepara o texto para a criptografia, mas também oferece uma representação visual das peças do cubo que seriam giradas durante a criptografia física, estabelecendo a base para o subsequente embaralhamento dos dados.
- Ao visualizar os dados alocados nas matrizes, o usuário pode obter uma compreensão mais clara de como o texto é segmentado e preparado para a criptografia, refletindo as rotações de um Cubo de Rubik real.

3.2 Método Insere

Basicamente essa função é chamada após o Aloca, foi montada baseada nas matrizes 3 x 3 e insere o caractere armazenado pelo Aloca em sua respectiva aba, de uma forma que fique visível ao usuário.

```
1 private void Insere() {
2     b1.Text = blue[0, 0];
```

```
3       b2.Text = blue[0, 1];
4       b3.Text = blue[0, 2];
5
6       b4.Text = blue[1, 0];
7       b5.Text = blue[1, 1];
8       b6.Text = blue[1, 2];
9
10      b7.Text = blue[2, 0];
11      b8.Text = blue[2, 1];
12      b9.Text = blue[2, 2];
13
14      o1.Text = orange[0, 0];
15      o2.Text = orange[0, 1];
16      o3.Text = orange[0, 2];
17
18      o4.Text = orange[1, 0];
19      o5.Text = orange[1, 1];
20      o6.Text = orange[1, 2];
21
22      o7.Text = orange[2, 0];
23      o8.Text = orange[2, 1];
24      o9.Text = orange[2, 2];
25
26      y1.Text = yellow[0, 0];
27      y2.Text = yellow[0, 1];
28      y3.Text = yellow[0, 2];
29
30      y4.Text = yellow[1, 0];
31      y5.Text = yellow[1, 1];
32      y6.Text = yellow[1, 2];
33
34      y7.Text = yellow[2, 0];
35      y8.Text = yellow[2, 1];
36      y9.Text = yellow[2, 2];
37  }
```

3.3 Método ColetaInfEAtTextoCrip

```
1       private void ColetaInfEAtTextoCrip() {
2           StringBuilder textoCripBuilder = new StringBuilder();
3           for (int grupo = 0; grupo < 3; grupo++) {
4               ColetaInfGrupo(textoCripBuilder, PegaMatrizIndex(grupo * 2),
                    PegaMatrizIndex(grupo * 2 + 1));
5           }
6           textoCrip.Text = textoCripBuilder.ToString();
7       }
```

```
8
9       private void ColetaInfGrupo(StringBuilder builder, string[,]
            matrizDireita, string[,] matrizEsquerda) {
10          for (int i = 0; i < 3; i++) {
11              AddInfStringBuilder(builder, matrizDireita, i, true);
12              AddInfStringBuilder(builder, matrizEsquerda, i, false);
13          }
14      }
15
16      private void AddInfStringBuilder(StringBuilder builder, string[,]
            matriz, int i, bool isMatrizDaDireita) {
17          for (int j = 0; j < 3; j++) {
18              if (isMatrizDaDireita) {
19                  builder.Append(matriz[i, j]);
20              }else{
21                  builder.Append(matriz[i, j]);
22              }
23          }
24      }
25
26      private string[,] PegaMatrizIndex(int index) {
27          switch (index) {
28              case 0: return blue;
29              case 1: return orange;
30              case 2: return yellow;
31              case 3: return green;
32              case 4: return red;
33              case 5: return white;
34              default: return null;
35          }
36      }
```

- O método ColetaInfEAtTextoCrip() e as funções auxiliares que ele invoca são a espinha dorsal da transformação do texto na aplicação.

- Decompondo o processo para entender o mecanismo por trás de cada passo e como eles se encaixam dentro do contexto maior da criptografia pode ser:

```
1       private void ColetaInfEAtTextoCrip() {
2           StringBuilder textoCripBuilder = new StringBuilder();
3           for (int grupo = 0; grupo < 3; grupo++) {
4               ColetaInfGrupo(textoCripBuilder, PegaMatrizIndex(grupo * 2),
5               PegaMatrizIndex(grupo * 2 + 1));
6           }
7           textoCrip.Text = textoCripBuilder.ToString();
8       }
```

3.4 Propósito Geral

- Projetado para concatenar os caracteres de várias matrizes de string (que representam as faces do Cubo de Rubik) em uma única string. É uma etapa essencial na preparação do texto criptografado para a visualização.

3.5 Inicialização do StringBuilder

- Um StringBuilder é instanciado para acumular e construir o texto criptografado.
- A escolha de StringBuilder é devido à sua eficiência em manipular e concatenar strings, principalmente em situações que exigem múltiplas operações de concatenação.

3.6 Loop de Concatenação

- Um loop for itera três vezes, cada uma correspondendo a um grupo de faces do Cubo de Rubik. Em cada iteração:
  - O método ColetaInfGrupo é chamado com o StringBuilder atual e um par de matrizes de string.
  - As matrizes são selecionadas pelo método PegaMatrizIndex, que recebe um índice e retorna a matriz correspondente. Este método utiliza um switch para mapear índices aos nomes das matrizes.

3.7 Construção Final do Texto

- Após percorrer todas as matrizes, o conteúdo acumulado no StringBuilder é convertido em uma string e atribuído à variável textoCrip.Text, que contém o texto criptografado final.

3.8 Funções Auxiliares

```
1     private void ColetaInfGrupo(StringBuilder builder, string[,]
2     matrizDireita, string[,] matrizEsquerda) {
3         for (int i = 0; i < 3; i++) {
4             AddInfStringBuilder(builder, matrizDireita, i, true);
5             AddInfStringBuilder(builder, matrizEsquerda, i, false);
6         }
7     }
8
```

```
9     private void AddInfStringBuilder(StringBuilder builder, string[,]
10    matriz, int i, bool isMatrizDaDireita) {
11        for (int j = 0; j < 3; j++) {
12            if (isMatrizDaDireita) {
13                builder.Append(matriz[i, j]);
14            } else {
15            builder.Append(matriz[i, j]);
16            }
17        }
18    }
```

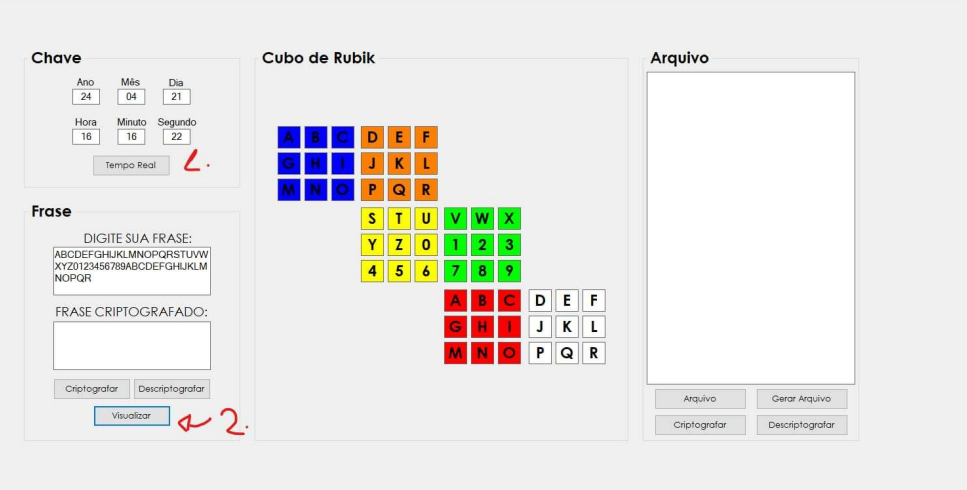
- Função ColetaInfGrupo(StringBuilder, string[,], string[,])
  - Encarrega-se de iterar pelas linhas das matrizes passadas como argumento e coletar os dados nelas contidos.
  - Para cada linha, chama o método AddInfStringBuilder duas vezes: uma para a matriz da direita (true) e outra para a matriz da esquerda (false).
- Função AddInfStringBuilder(StringBuilder, string[,], int, bool)
  - Responsável por adicionar ao StringBuilder as informações de uma linha específica da matriz.
  - Intera pelas colunas da linha especificada (i) e usa o método Append do StringBuilder para adicionar os caracteres ao texto final.
  - O parâmetro isMatrizDaDireita indica de qual matriz os caracteres devem ser obtidos.
- Função PegaMatrizIndex(int)
  - Retorna uma matriz específica baseada em um índice fornecido. É utilizada para abstrair a lógica de seleção da matriz e melhorar a clareza do código.
  - Usa uma instrução switch para determinar qual matriz retornar com base no valor de index.

```
1     private string[,] PegaMatrizIndex(int index) {
2         switch (index) {
3             case 0: return blue;
4             case 1: return orange;
5             case 2: return yellow;
6             case 3: return green;
7             case 4: return red;
8             case 5: return white;
9             default: return null;
10        }
11    }
```

– Com isso, o botão Visualizar está em funcionamento baseado na lógica que foi proposto inicialmente, fora que já há um caminho andado quando for realizado o procedimento de criptografia do texto.

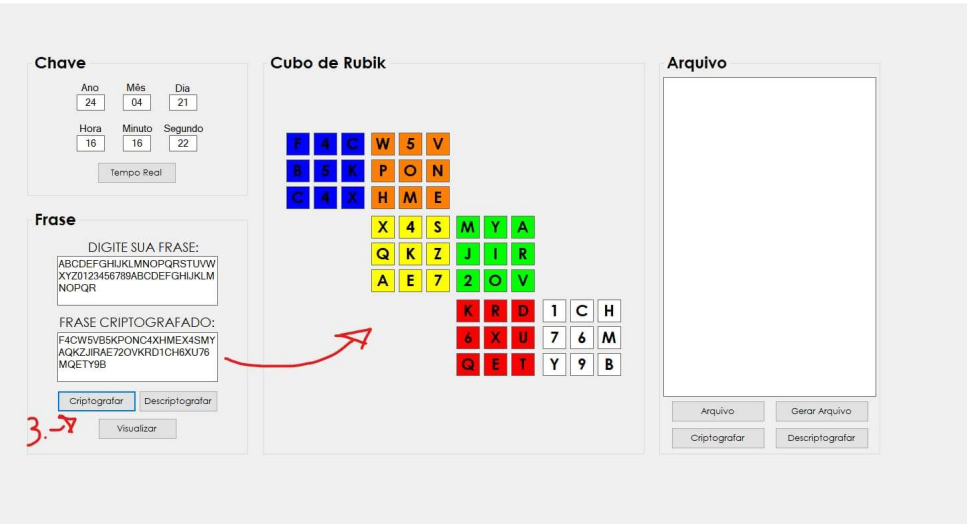
4 BOTÃO CRIPTOGRAFAR

Figura 4.1 – *Tutorial da criptografia*



Fonte: Autor

Figura 4.2 – *Texto criptografado*



Fonte: Autor

- O botão "Criptografar"atua como o principal gatilho para iniciar o processo de codificação do texto. A funcionalidade de criptografia é projetada para capturar a complexidade e o dinamismo dos giros do cubo, proporcionando assim uma saída única de texto baseada na chave de tempo - ou seja, a hora atual do sistema.

```
1     private void btn_Criptografar_Click(object sender, EventArgs e) {
2         Criptografar();
3         Insere();
4         ColetaInfEAtTextoCrip();
5     }
6     void Criptografar() {
7         int anos = Convert.ToInt32(Ano.Text);
8         int meses = Convert.ToInt32(Mes.Text);
9         int dias = Convert.ToInt32(Dia.Text);
10        int horas = Convert.ToInt32(Hora.Text);
11        int minutos = Convert.ToInt32(Min.Text);
12        int segundos = Convert.ToInt32(Sec.Text);
13        int Tempo = 1;
14        bool proximo = true;
15            while (proximo) {
16                proximo = false;
17                if (Tempo <= anos) {
18                Movimenta(blue);
19                Lado789789(red, white);
20                Lado147789(yellow, red);
21                Lado147147(orange, yellow);
22                Lado789147(aux, orange);
23                proximo = true;
24                }
25                Tempo++;
26            }
27        for (int I = 0; I < 3; I++) {
28            for (int J = 0; J < 3; J++) {
29                if (caesar.IndexOf(blue[I, J]) > -1)
30                blue[I, J] = caesar.Substring(caesar.IndexOf(blue[I,
31                J]) + anos, 1);
32            }
33        }
34    }
```

4.1 Função de Criptografia

- A função Criptografar é invocada ao clicar no botão "Criptografar". Ela captura os valores temporais - ano, mês, dia, hora, minuto e segundo - da interface do usuário e os converte de strings para números inteiros. Estes valores são utilizados para determinar a sequência e o número de movimentos ou "giros"aplicados às matrizes que representam as faces do Cubo de Rubik.

4.2 Lógica de Processamento

- O processo se inicia com um loop controlado pela variável booleana proximo. Em cada iteração do loop, verifica-se se o contador Tempo é menor ou igual a cada unidade de tempo (ano, mês, dia, etc.). Se for, as funções de movimentação do cubo, como Movimenta e Lado..., são chamadas para realizar giros específicos nas matrizes associadas às cores do Cubo de Rubik. A variável proximo é redefinida para true para permitir a próxima iteração do loop, simulando a continuação dos giros.

4.3 Movimentos do Cubo

- As funções de movimentação como Movimenta(blue) e Lado789789(red, white) são responsáveis por reorganizar os elementos dentro das matrizes. Cada função simula um tipo diferente de movimento do Cubo de Rubik.
- Por exemplo, Movimenta(blue) simboliza um giro da face azul do cubo, enquanto Lado789789 representa a transferência de elementos entre as faces vermelha e branca do cubo.

```
1    void Movimenta(string[,] face) {
2        for (int J = 0; J < 3; J++) {
3            for (int I = 0; I < 3; I++) {
4                aux[I, J] = face[I, J];
5            }
6        }
7        face[0, 2] = aux[0, 0];
8        face[1, 2] = aux[0, 1];
9        face[2, 2] = aux[0, 2];
10       face[0, 1] = aux[1, 0];
11       face[2, 1] = aux[1, 2];
12       face[0, 0] = aux[2, 0];
13       face[1, 0] = aux[2, 1];
14       face[2, 0] = aux[2, 2];
15   }
16
17   void Lado789789(string[,] origem, string[,] destino) {
18       for (int J = 0; J < 3; J++) {
19           for (int I = 0; I < 3; I++) {
20               aux[I, J] = destino[I, J];
21           }
22       }
23       destino[2, 0] = origem[2, 0];
24       destino[2, 1] = origem[2, 1];
25       destino[2, 2] = origem[2, 2];
```

```
26      }
27
28      void Lado147789(string[,] origem, string[,] destino) {
29          destino[2, 0] = origem[0, 0];
30          destino[2, 1] = origem[1, 0];
31          destino[2, 2] = origem[2, 0];
32      }
33
34      void Lado147147(string[,] origem, string[,] destino) {
35          destino[0, 0] = origem[0, 0];
36          destino[1, 0] = origem[1, 0];
37          destino[2, 0] = origem[2, 0];
38      }
39
40      void Lado789147(string[,] origem, string[,] destino) {
41          destino[0, 0] = origem[2, 0];
42          destino[1, 0] = origem[2, 1];
43          destino[2, 0] = origem[2, 2];
44      }
```

4.4 Atualização das Matrizes

- Após determinar os movimentos baseados na chave de tempo, o algoritmo procede para a substituição de caracteres dentro das matrizes. Utilizando a string caesar - uma referência à Cifra de César -, a substituição é feita indexando o caractere atual na matriz e encontrando o seu equivalente deslocado dentro da string caesar.

4.5 Finalização

- Concluído o processo de movimentação e substituição, o texto criptografado é coletado e consolidado no método ColetaInfEAtTextoCrip(), onde o StringBuilder é utilizado para concatenar os caracteres das matrizes transformadas em uma única string que representa o texto criptografado final.

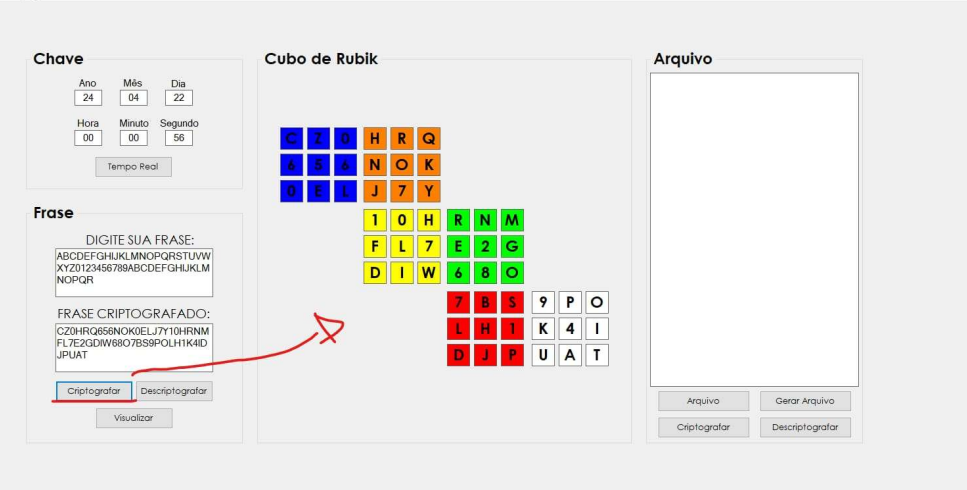
4.6 Movimentações Específicas

- Movimenta(string[,]):
  - Este método realiza uma série de trocas de posições dentro de uma matriz que simula um giro de uma face do cubo. As posições dos elementos são alteradas de maneira que reflete um movimento de 90 graus no Cubo de Rubik.
- Lado789789, Lado147789, Lado147147, Lado789147:

– Cada um desses métodos representa um movimento de peças entre duas faces do cubo, mantendo o paralelo com os movimentos reais do Cubo de Rubik.

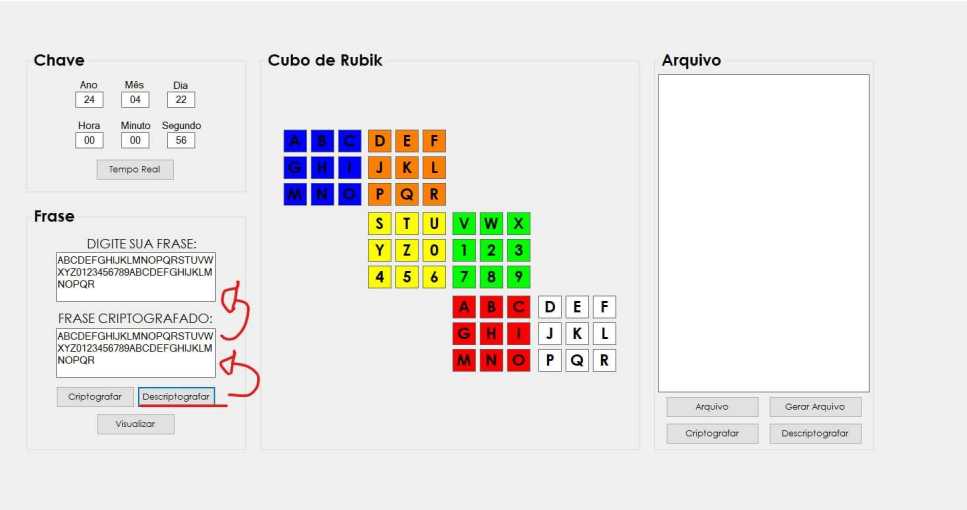
5 BOTÃO DESCRIPTOGRAFAR

Figura 5.1 – *Tutorial descriptografia*



Fonte: Autor

Figura 5.2 – *Texto descriptografado*



Fonte: Autor

- No coração da interface que simula a mecânica do Cubo de Rubik para criptografia, o botão "Descriptografar"é essencial para desvendar o texto codificado. Este botão ativa um processo que inverte os passos de criptografia, restabelecendo a mensagem original a partir do seu estado cifrado.

5.1 Método Descriptografar

```
1     void Descriptografar() {
2         int Tempo = 0;
3         int anos = Convert.ToInt32(Ano.Text);
4             if (anos > Tempo) {
5                 Tempo = anos;
6             }
7         int meses = Convert.ToInt32(Mes.Text);
8             if (meses > Tempo) {
9                 Tempo = meses;
10            }
11        int dias = Convert.ToInt32(Dia.Text);
12            if (dias > Tempo) {
13                Tempo = dias;
14            }
15        int horas = Convert.ToInt32(Hora.Text);
16            if (horas > Tempo) {
17                Tempo = horas;
18            }
19        int minutos = Convert.ToInt32(Min.Text);
20            if (minutos > Tempo) {
21                Tempo = minutos;
22            }
23        int segundos = Convert.ToInt32(Sec.Text);
24            if (segundos > Tempo) {
25                Tempo = segundos;
26            }
27        for (int I = 0; I < 3; I++) {
28            for (int J = 0; J < 3; J++) {
29                if (caesar.IndexOf(blue[I, J]) > -1)
30                blue[I, J] =
31                caesar.Substring(caesar.LastIndexOf(blue[I, J])
32                - anos, 1);
33            }
34        }
35        while (Tempo > 0) {
36            if (Tempo <= anos) {
37                MovimentaVL(blue);
38                LadoVL147147(yellow, orange);
39                LadoVL147789(red, yellow);
40                LadoVL789789(white, red);
41                LadoVL789147(aux, white);
42            }
43            Tempo--;
44        }
45    }
```

5.1.1 Objetivo

- O procedimento de descriptografia segue uma abordagem inversa à criptografia, realizando movimentos contrários aos aplicados inicialmente, utilizando a maior unidade de tempo como referência para a quantidade de inversões necessárias para que possa apresentar um bom funcionamento.

5.1.2 Determinação do Maior Valor Temporal

- O método identifica o maior valor entre os componentes de tempo inseridos (ano, mês, dia, hora, minuto e segundo). Esse valor será o ponto de partida para as iterações de descriptografia, garantindo que todas as permutações possam ser revertidas.

5.1.3 Substituição de Caracteres

- Utilizando a técnica LastIndexOf, o método procura a posição mais recente de um caractere na matriz em comparação com a string caesar. Então, subtrai o valor temporal correspondente para retroceder à posição original do caractere, antes da criptografia ter sido aplicada.

5.1.4 Loop de Inversão

- Um loop decremental começa a partir do maior valor de tempo, executando a série de movimentos reversos enquanto o contador ainda se encontra dentro dos limites de cada unidade de tempo (segundos, minutos, etc.).
- Cada iteração deste loop corresponde a uma tentativa de desfazer um "giro"do cubo feito durante a criptografia.

5.2 Loop de Inversão

```
1     void MovimentaVL(string[,] face) {
2         for (int J = 0; J < 3; J++) {
3             for (int I = 0; I < 3; I++) {
4                 aux[I, J] = face[I, J];
5             }
6         }
7         face[0, 0] = aux[0, 2];
8         face[0, 1] = aux[1, 2];
9         face[0, 2] = aux[2, 2];
10        face[1, 0] = aux[0, 1];
11        face[1, 2] = aux[2, 1];
```

```
12          face[2, 0] = aux[0, 0];
13          face[2, 1] = aux[1, 0];
14          face[2, 2] = aux[2, 0];
15      }
16
17      void LadoVL789789(string[,] origem, string[,] destino) {
18          destino[2, 0] = origem[2, 0];
19          destino[2, 1] = origem[2, 1];
20          destino[2, 2] = origem[2, 2];
21      }
22
23      void LadoVL147789(string[,] origem, string[,] destino) {
24          destino[0, 0] = origem[2, 0];
25          destino[1, 0] = origem[2, 1];
26          destino[2, 0] = origem[2, 2];
27      }
```

5.2.1 MovimentaVL(string[,])

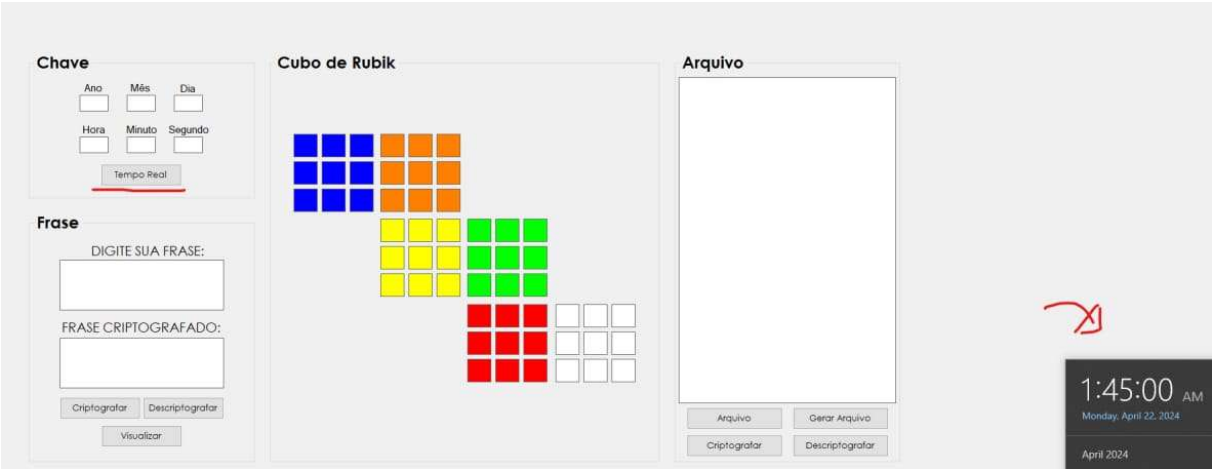
- Este método é a antítese do Movimenta, realocando os elementos dentro da matriz face para suas posições originais ou para posições intermediárias que gradualmente desfazem as rotações aplicadas durante a criptografia.

5.2.2 LadoVL... (LadoVL789789, LadoVL147789, etc.)

- Cada uma dessas funções é desenhada para desfazer uma transferência específica de dados entre matrizes. Elas realocam os elementos de volta às suas posições originais, desmanchando o padrão criado durante a codificação do texto.

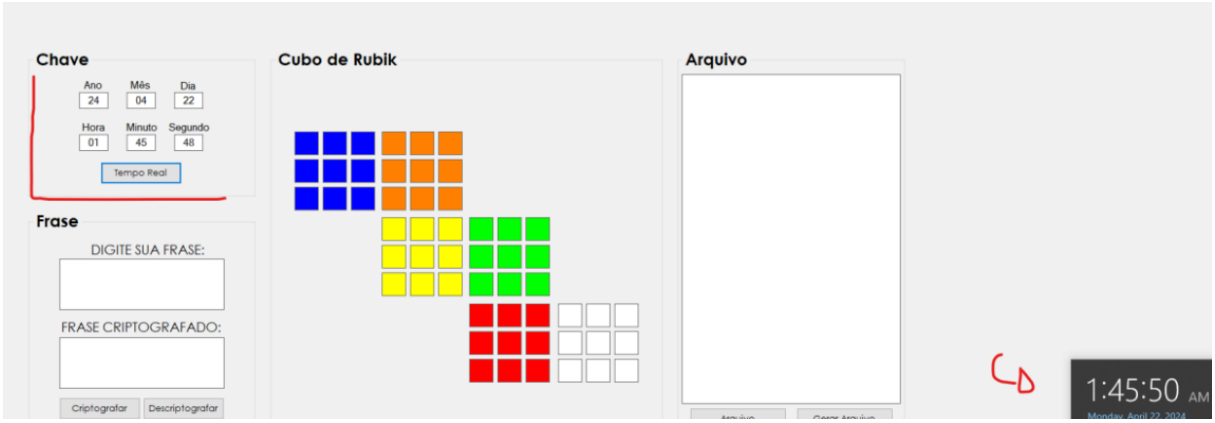
6 BOTÃO TEMPO REAL

Figura 6.1 – *Botão tempo real*



Fonte: Autor

Figura 6.2 – *Chave com o tempo atual da máquina*



Fonte: Autor

- Desempenha uma função crucial no estabelecimento da chave de criptografia, fornecendo valores de data e hora que são utilizados como parâmetros essenciais no processo de criptografia.

6.1 Método DataReal

```
1 private void DataReal() {
```

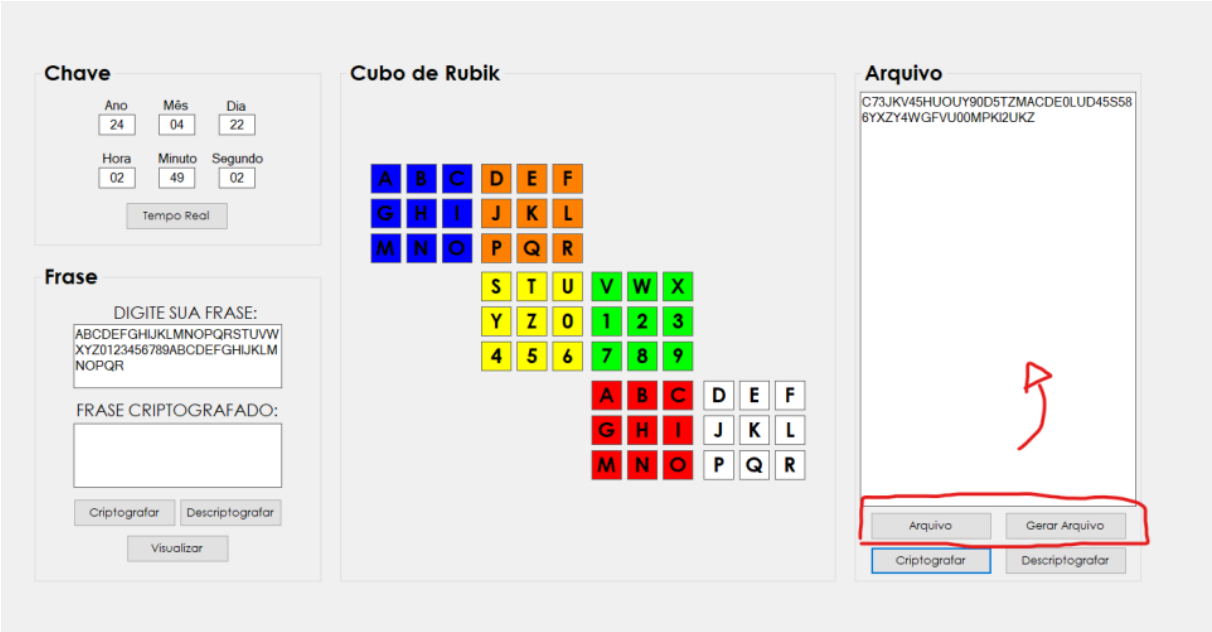
```
2       string AnoReal = DateTime.Now.ToString("yy");
3       string MesReal = DateTime.Now.ToString("MM");
4       string DiaReal = DateTime.Now.ToString("dd");
5       string HorReal = DateTime.Now.ToString("HH");
6       string MinReal = DateTime.Now.ToString("mm");
7       string SegReal = DateTime.Now.ToString("ss");
8       Ano.Text = AnoReal;
9       Mes.Text = MesReal;
10      Dia.Text = DiaReal;
11      Hora.Text = HorReal;
12      Min.Text = MinReal;
13      Sec.Text = SegReal;
14  }
15
16      private void button2_Click(object sender, EventArgs e) {
17          DataReal();
18      }
```

  - O método DataReal capta a data e hora correntes do sistema e as decompõe em seus componentes individuais - ano, mês, dia, hora, minuto e segundo. Estes componentes são, então, inseridos nos campos correspondentes da interface do usuário, preparando-os para serem utilizados como chave de criptografia.
  - A função DateTime.Now é utilizada para adquirir o momento exato atual.
  - Métodos ToString com diferentes especificadores de formato são aplicados para converter a data e hora atuais nos formatos requeridos.
  - Os valores obtidos são então transferidos para os campos de texto relevantes na interface do programa (Ano, Mes, Dia, Hora, Min, Sec), permitindo que sejam adotados como chave para a aplicação.

- Com essa ação, a interface do usuário é sincronizada com o relógio do sistema, e os valores de tempo são definidos de forma automática e precisa, fornecendo um elemento de dinamismo e segurança para o processo de criptografia, visto que a chave é sempre atual e exclusiva no momento de sua geração.

7 BOTÃO ARQUIVO E GERAR ARQUIVO

Figura 7.1 – *Geração de arquivo*



Fonte: Autor

- Esta fase é essencial, pois permite que o texto, uma vez criptografado ou descriptografado, possa ser tanto importado quanto exportado apenas clicando em simples botões.

7.1 Importação de Arquivo

- Funções que importam o arquivo:

```
1  private void button1_Click(object sender, EventArgs e) {
2      try {
3          using (OpenFileDialog openFileDialog = new OpenFileDialog()
               ) {
4          openFileDialog.Filter = "Arquivos de texto (*.txt)|*.txt";
5          openFileDialog.FilterIndex = 1;
6              if (openFileDialog.ShowDialog() == DialogResult.OK) {
7                  string nomeArquivo = openFileDialog.FileName;
8                  string conteudo = File.ReadAllText(nomeArquivo);
9                  ArquivoTXT.Text = conteudo;
10             }
11         }
12     } catch (Exception ex)
```

```
13 }
```

- O botão "Arquivo"ativa a função de importação, que invoca a janela de diálogo do sistema operacional.
- Esta janela, com seu filtro meticulosamente ajustado, direciona o usuário para uma navegação focada em documentos de texto, evitando distrações e facilitando a busca pelo arquivo desejado.
- A implementação do OpenFileDialog permite uma seleção intuitiva e acessível, encapsulada dentro de uma estrutura try-catch, que garante a resiliência do sistema contra eventuais erros de leitura de arquivos.
- Ao selecionar um arquivo e confirmar, o código procede para a leitura do conteúdo textual, carregando-o no campo ArquivoTXT da interface gráfica.
- Este processo de importação é feito de forma eficaz, permitindo ao usuário visualizar instantaneamente o conteúdo do arquivo dentro do contexto da aplicação.
- Em caso de falha durante a operação, uma caixa de diálogo alerta o usuário, informando a natureza do erro e evitando quaisquer interrupções abruptas ou perda de dados.

7.2 Gravar o arquivo

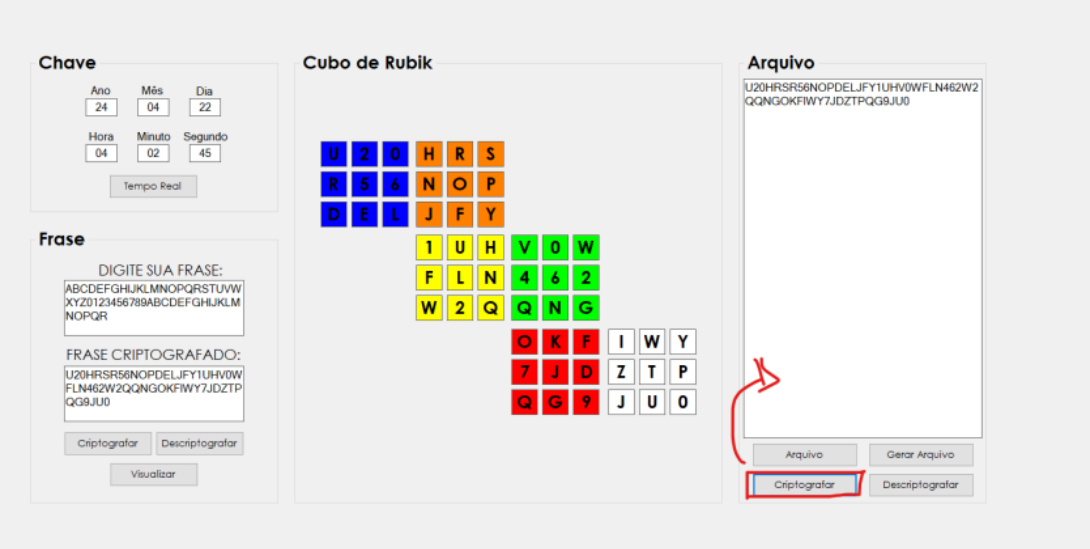
```
1  private void GerArq_Click(object sender, EventArgs e) {
2      string textoDoArquivo = ArquivoTXT.Text;
3      string informacoesExtras = $"\n\n Chave utilizada:
4      {Ano.Text}/{Mes.Text}/{Dia.Text} {Hora.Text}:{Min.Text}:{Sec.
           Text}";
5      textoDoArquivo += informacoesExtras;
6      SalvarArquivo(textoDoArquivo);
7  }
8
9  private void SalvarArquivo(string textoDoArquivo) {
10     try {
11         SaveFileDialog saveFileDialog = new SaveFileDialog();
12         saveFileDialog.Filter = "Arquivos de Texto (.txt)|.txt";
13         saveFileDialog.Title = "Salvar arquivo de texto";
14         saveFileDialog.InitialDirectory =
15         Environment.GetFolderPath(Environment.SpecialFolder.Desktop
               );
16         saveFileDialog.FileName = "MeuArquivo.txt";
17         if (saveFileDialog.ShowDialog() == DialogResult.OK) {
18             string filePath = saveFileDialog.FileName;
```

```
19                  using (StreamWriter sw = File.CreateText(filePath))
                        {
20                      sw.Write(textoDoArquivo);
21                  }
22              MessageBox.Show("Arquivo criado com sucesso!",
23              "Sucesso", MessageBoxButtons.OK, MessageBoxIcon.
                    Information);
24          }
25      } catch (Exception ex) {
26          MessageBox.Show("Ocorreu um erro ao criar o arquivo: " +
27          ex.Message, "Erro", MessageBoxButtons.OK, MessageBoxIcon.
                Error);
28      }
29  }
```

- O processo inicia-se com a ação do botão "Gerar Arquivo", que desencadeia a função "GerArq_Click".
- Aqui, o conteúdo textual presente no campo ArquivoTXT é capturado e enriquecido com informações adicionais que codificam os parâmetros usados na criptografia – especificamente, os dados de tempo.
- A variável textoDoArquivo é inicialmente atribuída ao conteúdo de um controle de texto ArquivoTXT.
- A concatenação da mensagem original com os detalhes da chave de criptografia cria um registro completo, que é então submetido ao método SalvarArquivo.
- Este método, por sua vez, apresenta uma caixa de diálogo SaveFileDialog para o usuário, permitindo que o destino do arquivo seja especificado de maneira intuitiva.
- A janela de diálogo é pré-configurada para sugerir a gravação de arquivos no formato texto (.txt), com o nome padrão "MeuArquivo.txt", e localizado no Desktop para fácil acesso.
- Durante o processo de salvamento, a abordagem de programação defensiva é mantida, envolvendo a operação em um bloco try-catch para antecipar e gerir eventuais falhas.
- Se o usuário prosseguir e confirmar o local de gravação, o texto é persistido no arquivo através da classe StreamWriter, que garante um gerenciamento de recursos eficiente ao encapsular a operação de escrita em um bloco using.
- Finalmente, o sucesso da gravação é comunicado ao usuário por meio de uma mensagem afirmativa. Em contrapartida, se houver um contratempo, o usuário é devidamente alertado por uma mensagem de erro, fechando o ciclo de feedback essencial para uma boa experiência do usuário.

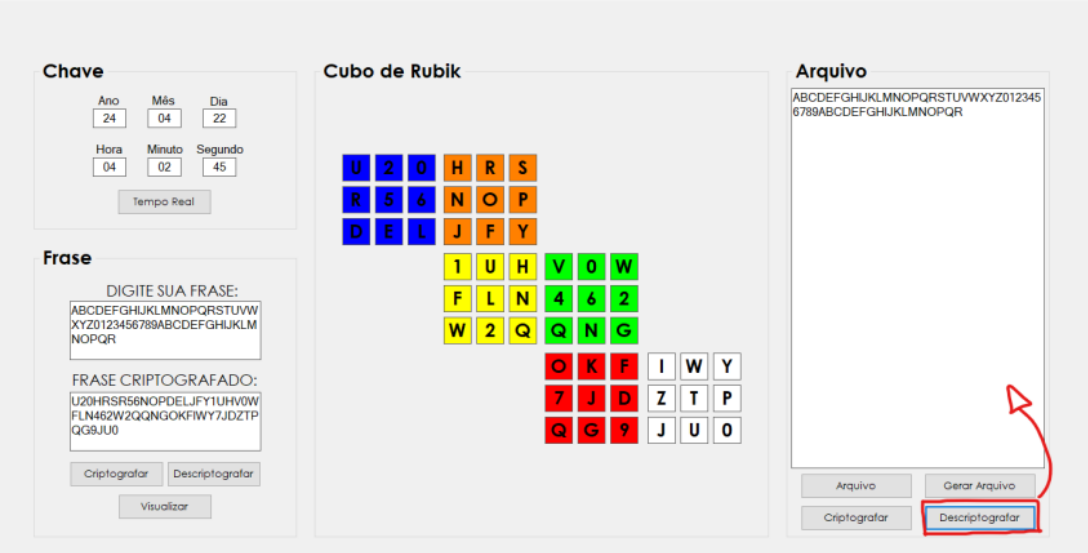
8 CRIPTOGRAFAR E DESCRIPTOGRAFAR O ARQUIVO DE TEXTO

Figura 8.1 – *Criptografia do arquivo*



Fonte: Autor

Figura 8.2 – *Descriptografia do arquivo*



Fonte: Autor

- A aplicação permite não apenas a criptografia e descriptografia de texto inserido diretamente na interface, mas também a leitura e gravação de conteúdo de e para arquivos de texto. Isso é viabilizado por duas funções distintas, detalhadas a seguir.

8.1 Criptografar Arquivo

```
1  private void ArquivoCrip_Click_1(object sender, EventArgs e) {
2      try {
3          string textoCompleto = ArquivoTXT.Text;
4          List<string> grupos = new List<string>();
5              for (int i = 0; i < textoCompleto.Length; i += 54) {
6                  grupos.Add(textoCompleto.Substring(i, Math.Min(54,
7                  textoCompleto.Length - i)));
8              }
9              if (grupos.Count > 0 && grupos[grupos.Count - 1].Length <
                   54) {
10                 int caracteresRestantes = 54 - grupos[grupos.Count - 1].
                       Length;grupos[grupos.Count - 1] +=
11                 "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
12                 ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
13                 ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".
14                 Substring(0, caracteresRestantes);
15             }
16             ArquivoTXT.Text = "";
17                 foreach (string grupo in grupos) {
18                     Aloca(grupo.Substring(0, 18), blue, orange);
19                     Aloca(grupo.Substring(18, 18), yellow, green);
20                     Aloca(grupo.Substring(36, 18), red, white);
21                     Criptografar();
22                     TxtArq();
23                 }
24     } catch (Exception ex) {
25         MessageBox.Show("Arquivo incompativel. Por favor selecione
               apenas arquivos do tipo 'TXT'" + ex.Message);
26     }
27
28 }
29
30 private void TxtArq() {
31     StringBuilder textoCripBuilder = new StringBuilder();
32         for (int grupo = 0; grupo < 3; grupo++) {
33             ColetaInfGrupo(textoCripBuilder, PegaMatrizIndex(grupo * 2),
34             PegaMatrizIndex(grupo * 2 + 1));
35         }
36         textoCripBuilder.AppendLine();
37         ArquivoTXT.AppendText(textoCripBuilder.ToString());
38 }
```

- O método ArquivoCrip_Click_1 é responsável por criptografar o texto importado. Ele executa as seguintes operações:

  - Divide o texto completo em segmentos de 54 caracteres, organizando-os em uma lista.
  - Verifica se o último segmento é menor que 54 caracteres, e em caso afirmativo, preenche o espaço restante com caracteres 'X'.
  - Limpa o texto atual da caixa de ArquivoTXT e inicia o processo de criptografar cada segmento, aplicando a lógica da criptografia de Rubik em subsequências de 18 caracteres.
  - Chama o método Criptografar() para cada segmento e, em seguida, o método TxtArq() para reconstruir o texto criptografado no controle ArquivoTXT.

- Após a execução, o texto no ArquivoTXT estará criptografado.

8.2 Descriptografar Arquivo

```
1 private void button2_Click_1(object sender, EventArgs e) {
2     string texto = ArquivoTXT.Text;
3     int numeroDeCaracteres = texto.Length;
4     int vezesChamada = numeroDeCaracteres / 54;
5     ArquivoTXT.Text = "";
6         for (int i = 0; i < vezesChamada; i++) {
7             int inicio = i * 54;
8             string grupo = texto.Substring(inicio, 54);
9             Aloca(grupo.Substring(0, 18), blue, orange);
10            Aloca(grupo.Substring(18, 18), yellow, green);
11            Aloca(grupo.Substring(36, 18), red, white);
12            Descriptografar();
13            TxtArq();
14        }
15 }
```

- A fase que restaura a mensagem original do conteúdo cifrado.
  - Armazena-se em texto o conteúdo criptografado disponível no controle ArquivoTXT.
  - Calcula-se a quantidade de segmentos de 54 caracteres contidos no texto criptografado através de numeroDeCaracteres e vezesChamada.
  - Prepara-se o campo de texto ArquivoTXT limpando seu conteúdo, para receber o texto após descriptografia.
  - Inicia-se um ciclo que processa cada segmento de 54 caracteres:
    * Calcula-se o índice inicial de cada segmento (inicio).

    - Cada segmento é extraído do texto cifrado (grupo).
    - Os segmentos de 18 caracteres são distribuídos entre as matrizes de cores que representam as faces do cubo de Rubik através do método Aloca.
    - nvoca-se o método Descriptografar() para aplicar a lógica inversa da criptografia sobre os segmentos e reverter as modificações.
    - O texto descriptografado é concatenado e apresentado no controle ArquivoTXT por meio da função TxtArq.

- Com esta metodologia, o algoritmo realiza a descriptografia efetivamente, convertendo o texto de volta para sua forma original, concluindo assim o ciclo completo de criptografia e descriptografia dentro da aplicação.

9 CONCLUSÕES

O projeto de criptografia de mensagens baseado no mapeamento do cubo de Rubik e implementado em C# oferece uma abordagem única e interessante para a segurança da informação. Ao utilizar o mapeamento do cubo de Rubik como base para a criptografia, o projeto aproveita as propriedades complexas e aparentemente caóticas do cubo para criar um método de criptografia robusto e desafiador.

A utilização da hora e data atual do computador do usuário como parte do processo de criptografia adiciona uma camada adicional de complexidade e segurança ao algoritmo. Isso significa que, mesmo que a chave de criptografia seja descoberta, a mensagem criptografada ainda estará protegida por meio da dependência do tempo.

Além disso, a implementação em C# oferece uma linguagem de programação amplamente utilizada e suportada, o que facilita a integração e a manutenção do sistema. A combinação desses elementos torna o projeto não apenas desafiador e interessante do ponto de vista técnico, mas também potencialmente útil para aplicações práticas que exigem alto nível de segurança na comunicação de dados. No entanto, é importante garantir que o algoritmo seja cuidadosamente projetado e testado para garantir sua eficácia e resistência a ataques.

REFERÊNCIAS BIBLIOGRÁFICAS

1. Leitura e salvamento do arquivo: https://learn.microsoft.com/pt-br/troubleshoot/developer/visualstudio/csharp/language-compilers/read-write-text-file
2. Sobre tipos de criptografia e sua importância: https://shorturl.at/dquY9
3. Inspiração para o método de criptografia: https://pt.wikipedia.org/wiki/Cifra-de-César
4. Código de pesquisa por pastas: https://youtu.be/PP5QkpUDak0?feature=shared