Revista Eletrônica
Paulista de Matemática

ISSN 2316-9664
v. 23, n. 1, jul. 2023
Artigo de Pesquisa

Criptografia simétrica com uma função afim

Simmetric-key cryptography using a linear function

Jose Laudelino de Menezes Neto
Campus IV - Rio Tinto
Universidade Federal da Paraíba
laudelino@dcx.ufpb.br

Welisson Martins Mota
Campus I - João Pessoa
Universidade Federal da Paraíba
professorwelisson1@gmail.com

Resumo

Criptografia é o procedimento de codificar um texto com o intuito de o tornar não compreensível para pessoas não autorizadas para o ler. O objetivo deste artigo é apresentar um tipo de criptografia simétrica, utilizando congruências e uma função afim do tipo $f(x) = ax+b$. Indo além da teoria, ensinaremos como implementar este procedimento em um programa para criação de jogos chamado Construct, onde será possível criptografar e descriptografar mensagens. Auxiliado pelo Algoritmo Euclidiano, também criamos um programa para gerar chaves de codificação e decodificação. Esperamos que os procedimentos aqui apresentados, sirvam de modelo para o uso na prática, em sala de aula, auxiliando na motivação e ensino dos conceitos matemáticos envolvidos, bem como no estímulo do uso de novas tecnologias e interdisciplinaridade entre ciências da computação e matemática, devido a implementação computacional.

Palavras-chave: Algoritmo Euclidiano. Cifra de César. Matemática Aplicada.

Abstract

Cryptography is a method to code a text to let it in a non-comprehensive way to non-authorized people. In this paper, our objective is to show a kind of symmetric-key cryptography using modular arithmetic and a linear function $f(x) = ax + b$. We go beyond theory, implementing the procedure in a program to create games, called Construct. With this implementation, it will be possible to cryptography a message. Also, using Euclid's algorithm, we made another program to create cipher and decipher keys. We hope that the procedures presented here, can be used as a source to be applied in a classroom, being a way to motivate the students to understand the mathematical subjects presented, and also to stimulate the use of new technologies, as the implementation made in a computational program.

Keywords: Euclidean algorithm. Caesar cipher. Applied Mathematics.

Artigo recebido em out. 2022 e aceito em fev. 2023

1 Introdução

A criptografia remonta de muitos anos atrás, sendo utilizada desde a época do império romano, em meados do ano 100 antes de Cristo, por Julio César para transmitir textos secretos (GANASSOLI; SCHANKOSKI, 2015; LEMOS, 2010; MIRANDA; PAULA, 2021). No decorrer do tempo, vem se aprimorando, sendo bastante utilizada nos dias atuais na comunicação, principalmente nos aplicativos de mensagens instantâneas, como o Whatsapp.

Em linhas gerais, a criptografia consiste em pegar uma mensagem como "OLÁ COMO VAI VOCÊ" e cifrá-la em um texto incompreensível do tipo "EÓQYSEMEYÃ ÁYÃESV", de modo que apenas pessoas autorizadas tenham como recuperar a mensagem original.

Apresentaremos um tipo de criptografia simétrica e sua implementação de forma computacional, utilizando um programa online de criação de jogos, chamado Construct.

A criptografia simétrica, apesar de mais simples, mostra de forma sucinta como todo o procedimento criptográfico funciona, desde a escolha de um alfabeto, até a utilização de congruências de números inteiros.

Utilizamos uma função afim, do tipo $f(x) = ax + b$ para criptografar as mensagens, com a e b números inteiros, para mais detalhes consultar (MENEZES NETO, 2021). Os valores a e b fazem o papel da chave de criptografia simétrica. O nome simétrico vem da simetria entre a chave que criptografa, com a chave que descriptografa, e vice-versa. Em outras palavras, tendo a chave criptográfica, é teoricamente fácil obter a chave que descriptografa.

O procedimento de criptografia, aqui apresentado, é executado da seguinte forma. Dado um caractere **X** de um alfabeto, escrito na mensagem original, transformamos este caractere em um valor inteiro x e aplicamos a função $f(x) = ax + b = y$. Em seguida, com este valor y, utilizamos congruência de números inteiros, para associar este valor y a um caractere **Y** do alfabeto e ter o texto cifrado. Para descriptografar a mensagem **Y**, aplicamos a função inversa $f^{-1}(y) = a'y + b' = x$, recuperando o valor x e, consequentemente, o caractere da mensagem original **X**.

Este procedimento de criptografia, inclusive, foi sugerido como método de atividade em sala de aula de um modo mais simples, utilizando números reais, e com intuito de motivar o ensino de funções afins, conforme visto em (MIRANDA; PAULA, 2021). Aqui, a ineditidade cabe a interdisciplinaridade, envolvendo matemática e computação, e a apresentação na implementação do método criptográfico, usando congruências, no aplicativo Construct.

2 Fundamentação teórica

2.1 Escolha de um alfabeto

Para iniciar uma criptografia, devemos combinar com todos os usuários um alfabeto para escrever os textos. Ou seja, determinar quais caracteres podem ser utilizados e restringir todos os textos escritos a estes símbolos (GANASSOLI; SCHANKOSKI, 2015; LEMOS, 2010; MENEZES NETO, 2021).

Vamos nos limitar as letras maiúsculas de A a Z, alguns caracteres especiais de acentuação, além de permitir a utilização do espaço em branco, que iremos representar pelo símbolo de *underline* "_". Cada caractere deve ser associado a um valor numérico inteiro. Faremos a associação dada abaixo em (1).

$$\begin{array}{cccccccccccc} & & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} & \text{H} & \text{I} & \text{J} & \text{K} & \text{L} \\ & & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \text{M} & \text{N} & \text{O} & \text{P} & \text{Q} & \text{R} & \text{S} & \text{T} & \text{U} & \text{V} & \text{W} & \text{X} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ \text{Y} & \text{Z} & \_ & \tilde{\text{A}} & \text{Á} & \hat{\text{A}} & \text{É} & \hat{\text{E}} & \tilde{\text{O}} & \text{Ó} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 \end{array} \tag{1}$$

Ao todo, temos um total de 34 caracteres, contados a partir do número zero, até o número 33. Por isso, na implementação da criptografia, trabalharemos com congruência módulo 34, em símbolos $\equiv \pmod{34}$. De modo geral, se tivéssemos um total de m caracteres, deveríamos utilizar congruência módulo m, $\equiv \pmod{m}$.

Observamos a não inclusão da acentuação das vogais I e U, nem o uso do Ç, no nosso alfabeto, entretanto estes caracteres podem ser acrescentados. Em linhas gerais, por exemplo, para adicionar o caractere pretendido, digamos Ç, basta associá-lo a um número não utilizado, no caso o número 34, Ç $\leftrightarrow$ 34, porém, ao invés de trabalhar com módulo 34, passamos a trabalhar com módulo 35, visto que ampliamos a quantidade de caracteres. Para maiores detalhes recomendamos a leitura de (MENEZES NETO, 2021). O mesmo ocorre se for necessário acrescentar as letras em minúsculo, os algarismos de 0 a 9, ou outros caracteres.

2.2 Congruência de números inteiros

Uma das áreas de estudo da Teoria dos Números são as congruências módulo m, tem fundamental importância no processo de checagem da divisibilidade de um número por outro e, como vemos, tem aplicação nos sistemas de criptografia.

Dados x, y e m números inteiros, com $m > 1$, dizemos que x é congruente a y módulo m, em notação $x \equiv y \pmod{m}$, se x e y deixam mesmos restos quando divididos por m. É equivalente dizermos ainda, $x \equiv y \pmod{m}$ quando m divide $x - y$, em símbolos $m \mid x - y$.

Em uma linguagem mais computacional, temos $x \equiv y \pmod{m}$ se, e somente se, $x\%m$ é igual a $y\%m$. Então, passaremos a utilizar o comando $\%m$ no cálculo de congruências módulo m.

Por exemplo, $32 \equiv 4 \pmod{7}$, ou seja, $32\%7$ é igual a $4\%7$.

Observação 1 *A simbologia $b \mid a$, "b divide a", significa que o resto da divisão de a por b é igual a zero, exemplo:* $2 \mid 4$. *Já $b \nmid a$, "b não divide a", quer dizer que a quando dividido por b não deixa resto zero, exemplo* $3 \nmid 7$.

2.3 Algorítmo Euclidiano e mdc

O Algorítmo Euclidiano é um método matemático utilizado para calcular o Máximo Divisor Comum (mdc) entre dois números inteiros (HEFEZ, 2016; LEMOS, 2010; MENEZES NETO, 2021).

Sejam $a, b \in \mathbb{Z}$. Dizemos que $d \geq 0$ é um mdc de a e b, em símbolos $d = \text{mdc}(a, b)$, se

1. $d \mid a$ e $d \mid b$; e
2. se $c \in \mathbb{Z}$ é tal que $c \mid a$ e $c \mid b$ então $c \mid d$.

Se o mdc existe, então ele é único. O Teorema 2 abaixo garante a existência do mdc entre dois números inteiros e sua demonstração nos dá o Algoritmo Euclidiano para o cálculo de mdc. Antes de enunciar o Teorema 2, precisamos do seguinte Lema.

Lema 1 *Sejam $a, b, n \in \mathbb{Z}$. Se existir o $mdc(a, b - n \cdot a)$, então o $mdc(a, b)$ existe, e ainda, $mdc(a, b) = mdc(a, b - n \cdot a)$.*

Demonstração. Consultar referência (HEFEZ, 2016). ■

Teorema 2 *Para todo $a, b \in \mathbb{Z}$, existe $mdc(a, b) = d$.*

Demonstração. É equivalente mostrar para $a, b \in \mathbb{N}$, pois existe uma propriedade que garante $\mathrm{mdc}(a, b) = \mathrm{mdc}(|a|, |b|)$ (HEFEZ, 2016), e ainda, como $\mathrm{mdc}(a, b) = a$ quando $a \mid b$, nos resta analisar o caso em que $1 < a < b$ e $a \nmid b$. Pois bem, aplicamos o algoritimo da divisão, obtendo

$$b = a \cdot q_1 + r_1, \text{ com } 0 < r_1 < a.$$

Assim, pelo Lema 1, temos que se $r_1 \mid a$, então

$$\mathrm{mdc}(a, b) = \mathrm{mdc}(a, b - a \cdot q_1) = \mathrm{mdc}(a, r_1) = r_1.$$

Se $r_1 \nmid a$, temos

$$a = r_1 \cdot q_2 + r_2 \text{ com } 0 < r_2 < r_1 \quad (r_2 = a - r_1 \cdot q_2)$$

Logo, se $r_2 \mid r_1$, então,

$$\begin{aligned} \mathrm{mdc}(a, b) &= \mathrm{mdc}(a, r_1) \quad \text{e} \\ \mathrm{mdc}(a, r_1) &= \mathrm{mdc}(a - r_1 \cdot q_2, r_1) = \mathrm{mdc}(r_2, r_1) = r_2. \end{aligned}$$

Se $r_2 \nmid r_1$, tem-se

$$r_1 = r_2 \cdot q_3 + r_3 \text{ com } 0 < r_3 < r_2 \quad (r_3 = r_1 - r_2 \cdot q_3).$$

Continuamos o procedimento até parar, que é o momento onde encontramos o mdc. Note que essa parada sempre ocorre, do contrário teríamos uma sequência de números naturais $a > r_1 > r_2 > \ldots$ que não possui menor elemento, o que, pelo princípio da boa ordenação, é um absurdo. Portanto, para algum k, temos que $r_k \mid r_{k-1}$, o que nos dá $\mathrm{mdc}(a, b) = r_k$.

De modo simples, podemos dizer que o $\mathrm{mdc}(a, b) = r_k$, onde r_k é o último resto não nulo das divisões sucessivas elencadas anteriormente. ■

Teorema 3 *Seja d o mdc de dois inteiros a e b, então existem inteiros t e s tais que $a \cdot t + b \cdot s = d$*

Demonstração. Consultar referência (HEFEZ, 2016). ■

Este Teorema 3 é de suma importância quando tivermos de calcular o inverso multiplicativo de um número inteiro em congruências módulo $m \in \mathbb{Z}_+^*$, conforme destacamos na Observação 4 a seguir.

Observação 4 *Caso* $mdc(a, m) = 1$*, então existem* $t, s \in \mathbb{Z}$ *tais que* $a \cdot t + m \cdot s = 1$*, ou seja,*

$$m \mid a \cdot t - 1 \quad \Rightarrow \quad a \cdot t \equiv 1 (mod\, m),$$

o que nos leva a ver que t *é o inverso multiplicativo de* a *em congruência módulo* m*.*

Exemplo 5 *Determine o valor de* d*, tal que* $mdc(551, 874) = d$ *e calcule os inteiros* t *e* s *tais que*

$$551 \cdot t + 874 \cdot s = d.$$

Solução. Aplicando o algoritimo Euclidiano, temos

$$\begin{aligned} 874 &= 551 \cdot 1 + 323 \\ 551 &= 323 \cdot 1 + 288 \\ 323 &= 228 \cdot 1 + 95 \\ 228 &= 95 \cdot 2 + 38 \\ 95 &= 38 \cdot 2 + \mathbf{19} \\ 38 &= 19 \cdot 2 + 0. \end{aligned} \tag{2}$$

Logo, o $\text{mdc}(551, 874) = 19$. Neste caso, sendo o mdc diferente de 1, implica que 551 não tem inverso multiplicativo em congruência módulo 874.

Arrumando as equações em (2) obtemos

$$551 \cdot (-19) + 874 \cdot 12 = 19,$$

isto significa que determinamos $t = -19$ e $s = 12$.

Exemplo 6 *Determine o valor de* d*, tal que* $mdc(13, 34) = d$ *e calcule os inteiros* t *e* s *tais que*

$$13 \cdot t + 34 \cdot s = d.$$

Solução. Fazemos o mesmo procedimento do Exemplo anterior, aplicamos o algoritimo Euclidiano. Assim, obtemos

$$\begin{aligned} 34 &= 13 \cdot 2 + 8 \\ 13 &= 8 \cdot 1 + 5 \\ 8 &= 5 \cdot 1 + 3 \\ 5 &= 3 \cdot 1 + 2 \\ 3 &= 2 \cdot 1 + \mathbf{1} \\ 2 &= 1 \cdot 2 + 0. \end{aligned} \tag{3}$$

Portanto, o $\text{mdc}(13, 34) = 1$

Organizando as equações em (3) temos

$$13 \cdot (-13) + 34 \cdot 5 = 1,$$

ou seja, encontramos $t = -13$ e $s = 5$ atendendo ao que é enunciado no Teorema 3. Além disso, como o mdc foi igual a 1, de acordo com a Observação 4, temos que o inverso de 13 em congruência módulo 34 é igual a 21, pois $21 \equiv -13 (\text{mod } 34)$.

Resolução de mais exemplos deste tipo dos Exemplos 5 e 6 são encontrados em (MENEZES NETO, 2021).

3 Implementação no Construct

O Construct é um aplicativo para criação de jogos, desenvolvido pela empresa Scirra LTD. Faremos a implementação da criptografia neste programa. O intuito é o de utilizar a interface gráfica disponibilizada e deixar o procedimento mais amigável.

Outra vantagem do Construct é o fato de ser todo online, rodando direto no navegador de internet, sem a necessidade de instalação. Inclusive, o Construct funciona na maioria dos dispositivos móveis, como celulares e tablets. A programação do Construct é estruturada por meio de *Eventos* ao invés de linhas de código. Também sendo esta outra premissa do Construct, ensinar e familiarizar o usuário com programação, com a opção de utilizar ou não linhas de comando e o uso dos "IF", "THEN" e "ELSE".

O cadastro no site garante um número maior de eventos. Sem fazer o cadastro no site do Construct, somos limitados a um total de 25 eventos, já com conta cadastrada e verificada, suporta um total de 50 eventos. Para as implementações a serem feitas aqui neste texto, não será necessário cadastro no site.

O editor do Construct é bem intuitivo com muitas instruções disponíveis em tela e o seu acesso se dá através do seguinte endereço: `https://editor.construct.net`.

3.1 Procedimento da criptografia

Nesta subseção, explicaremos o procedimento executado no programa que iremos implementar no Construct. Faremos um programa que criptografa, e descriptografa, mensagens utilizando uma função afim $f(x) = ax + b$ e congruência módulo m. O programa é estruturado na forma a seguir.

Passo 1: Usuário escreve o texto a ser criptografado, digamos "OI" e escolhe a chave de criptografia, os valores a e b, que correspondem a função $f(x) = ax + b$. Como nosso alfabeto está limitado aos caracteres dados em (1), um total de 34 caracteres, então a e b devem ter valores entre os inteiros de 0 a 33, com a restrição de que $\text{mdc}(a, 34) = 1$, isso para garantir a existência da função inversa $f^{-1}(x) = a'x + b'$ em congruência módulo 34, pois a' é o inverso multiplicativo de a em congruências módulo m, ou seja, $a' \cdot a \equiv 1 (\text{mod } m)$, veja Observação 4, e $b' \equiv -a' \cdot b (\text{mod } m)$, mais detalhes em (MENEZES NETO, 2021). A título de explicação, escolhemos $a = 13$ e $b = 26$, notamos que $\text{mdc}(13, 34) = 1$.

Passo 2: Cada caractere da mensagem, no caso "OI", é separado e associado com seu respectivo valor numérico dado em (1). Caractere "O" associa com valor 14, e "I" com valor 8. Em seguida, nestes valores, o programa aplica a função $f(x) = ax + b$, $a = 13$ e $b = 26$, $f(14) = 208$ e $f(8) = 130$.

Passo 3: Para obter a mensagem criptografada, o programa utiliza congruências módulo 34 utilizando o comando %34 nos valores 208 e 130. No caso, é feito 208%34 = 4 e 130%34 = 28, equivalente a dizer, respectivamente, $208 \equiv 4 (\text{mod } 34)$ e $130 \equiv 28 (\text{mod } 34)$. Assim, associamos os valores 4 e 28 com seus respectivos caracteres em (1), obtendo a mensagem cifrada "EÁ". Ou seja, o programa criptografou "OI" na mensagem cifrada "ÉA".

Implementamos estes três passos no Construct.

3.2 Criptografando no Construct

Iniciamos um novo projeto no Construct. Nesta primeira etapa, criamos a interface gráfica do programa na aba *Layout 1*, damos um duplo clique na tela para adicionar itens necessários: quatro itens *Entrada de texto*, sendo dois com o nome padrão dado pelo próprio sistema *EntradaDeTexto* e *EntradaDeTexto2*, e dois deles alteramos o nome para *valorA* e *valorB* com a opção *Tipo* definida para *Número*, e adicionamos um item *Botão* (veja Fig. 1).

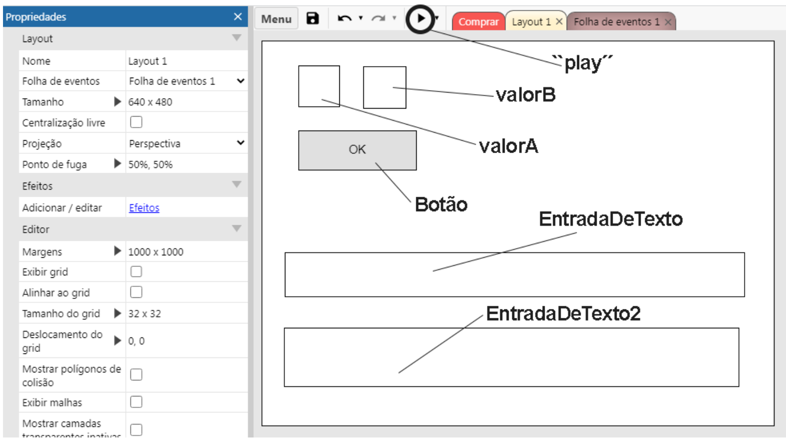

Figura 1: *Layout 1*

Na aba *Folha de eventos 1* é onde implementamos os **Passos 1**, **2** e **3** descritos na Subseção 3.1. Criamos cinco *variáveis globais*, três do tipo *número*, nomeadas como **A**, **B** e **TotalCarac**, e duas do tipo *string* **TextoEntrada** e **TextoSaida**. A variável **TotalCarac** deve ter seu valor setado para a quantidade total de caracteres do alfabeto utilizado, no caso, 34.

Adicionamos um primeiro evento e escolhemos o *Botão* e a opção *Ao clicar*. Neste evento, adicionamos as *ações* de *Sistema* para *Definir valor* as variáveis globais

• a variável **A** definida para *valorA.texto*, está definindo o valor a digitado pelo usuário da função $f(x) = ax + b$;

• a variável **B** definida para *valorB.texto*, está definindo o valor a digitado pelo usuário da função $f(x) = ax + b$;

• **TextoEntrada** setado para *EntradaDeTexto.texto*, está definindo o texto digitado pelo usuário a ser criptografado; e

• **TextoSaída** definido para “ ”, está definindo o texto a ser exibido no término do procedimento criptográfico, por enquanto, nada a ser exibido.

Este primeiro evento equivale ao **Passo 1** da Subseção 3.1 (veja Fig. 2).

Clicamos com o botão direito do mouse neste único evento e adicionamos um *sub-evento*. Neste *sub-evento*, criamos três variáveis locais, duas do tipo número **TempValor** e **TempCripto**, e uma do tipo *string* **TempLetra**. Ainda neste *sub-evento* adicionamos a condição de *Sistema* chamada de *Para*, equivalente a um comando *For*, na opção *Nome* escrevemos *“ind”*, com índice inicial zero e índice final com o seguinte comando *len(**TextoEntrada**)-1*. Este comando *len* conta a quantidade de caracteres do texto escrito pelo usuário com o objetivo de separar todos os caracteres para fazer a associação com cada valor numérico. Adicionamos a ação para definir a variável **TempLetra** com o comando *meio(TextoEntrada,loopindex,1)*, onde este comando a cada loop do *Para*, pega uma única letra da *string* **TextoEntrada**.

Feito isso, adicionamos neste sub-evento a opção *Adicionar script*, habilitando uma caixa de texto para digitação do script. Neste script, transformaremos em cada loop do *Para*, cada letra **TempLetra**, da variável **TextoEntrada**, no seu valor numérico, **TempValor**, dado pela associação em (1). O comando de script abaixo associa as letras A até D com seu respectivo número, porém o procedimento deve ser feito para todos os 34 caracteres em (1).

```
 if (localVars.TempLetra == "A") localVars.TempValor = 0;
else if (localVars.TempLetra == "B") localVars.TempValor = 1;
else if (localVars.TempLetra == "C") localVars.TempValor = 2;
else if (localVars.TempLetra == "D") localVars.TempValor = 3;
```

Após toda essa associação das letras ao seu respectivo valor numérico, aplicamos a função $f(x) = ax + b$ neste valor e usamos a congruência módulo 34. Ainda no mesmo script, isto equivale ao comando abaixo.

```
localVars.TempCripto = (runtime.globalVars.A*localVars.TempValor + runtime.globalVars.B) % runtime.globalVars.TotalCarac;
```

Observamos a utilização do comando % para calcular a congruência de números inteiros. Cabe destacar que até esta etapa, nestes três últimos parágrafos, equivale a descrição dada no **Passo 2**, da Subseção 3.1 (veja Fig. 2).

O valor numérico da letra **TempLetra** já criptografada é dado pela variável **TempCripto**. A próxima etapa é tranformar este valor numérico de volta ao caractere do alfabeto exibido em (1). Para tanto, ainda no mesmo script, adicionamos as linhas de comando para todos os valores de 0 a 33, associando o seu caractere. O comando abaixo faz esta associação até o valor 3.

```
 if (localVars.TempCripto == 0) runtime.globalVars.TextoSaida runtime.globalVars.TextoSaida + "A";
else if (localVars.TempCripto == 1) runtime.globalVars.TextoSaida runtime.globalVars.TextoSaida + "B";
else if (localVars.TempCripto == 2) runtime.globalVars.TextoSaida runtime.globalVars.TextoSaida + "C";
else if (localVars.TempCripto == 3) runtime.globalVars.TextoSaida runtime.globalVars.TextoSaida + "D";
```

Durante esta associação, vamos construindo a cada loop do *Para*, o texto de saída dado pela variável do tipo *string* **TextoSaida**. O procedimento descrito neste parágrafo equivale ao **Passo 3** da Subseção 3.1 (veja Fig. 2).

Ao final deste script, no *sub-evento* do *Para*, adicionamos um novo *sub-evento* em branco, apenas com uma ação no *EntradaDeTexto2* com a opção *Definir texto* ajustada para variável **TextoSaida**, para exibir o texto criptografado.

Toda a *Folha de eventos 1* é exibida na Fig. 2.

3.2.1 Funcionamento do programa

Para testarmos o programa, clicamos no "play" posicionado no topo da tela (Fig. 1). O programa já em execução, escrevemos o texto a ser cifrado, ou decifrado, na caixa de texto *EntradaDeTexto* e escolhemos os valores da chave de criptografia a e b nas caixas de texto intituladas, respectivamente, *valorA* e *valorB*.

O texto "OLÁ COMO VAI VOCÊ", usando $a = 13$ e $b = 26$, clicamos no *Botão* e o texto é cifrado na mensagem "EÓQYSEMEYÃ ÁYÃESV" (Fig. 3).

Para decifrar o texto, basta escrever o texto cifrado na caixa de texto *EntradaDeTexto* e utilizar para *valorA* e *valorB*, respectivamente, a' e b' tal que $f^{-1}(x) = a'x + b'$.

Escrevendo o texto cifrado "EÓQYSEMEYÃ ÁYÃESV" na caixa de texto *EntradaDeTexto* e utilizando $a = 21$ e $b = 32$, recuperamos a mensagem original.

Figura 2: *Folha de eventos 1*.

3.3 Implementação do Algoritmo Euclidiano

Faremos uma implementação do Algoritmo Euclidiano no Construct. O intuito é o de obter facilmente funções $f(x) = ax + b$ que possuem inversa em congruência módulo m e, de forma automática, ter a inversa $f^{-1}(x) = a'x + b'$. O usuário escreve os valores de m, a e b e o programa informa se $\text{mdc}(a, m) = 1$, em caso afirmativo, retorna os valores de a' e b'.

Em outras palavras, este programa é um facilitador na obtenção de chaves para nosso sistema criptográfico. O usuário digita $m = 34$, afinal trabalhamos com 34 caracteres, os valores da chave de codificação a e b, e o programa responde, quando existir, quais são as chaves de decodificação a' e b'.

Assim como feito na criação do programa de criptografia, elaboramos uma interface gráfica.

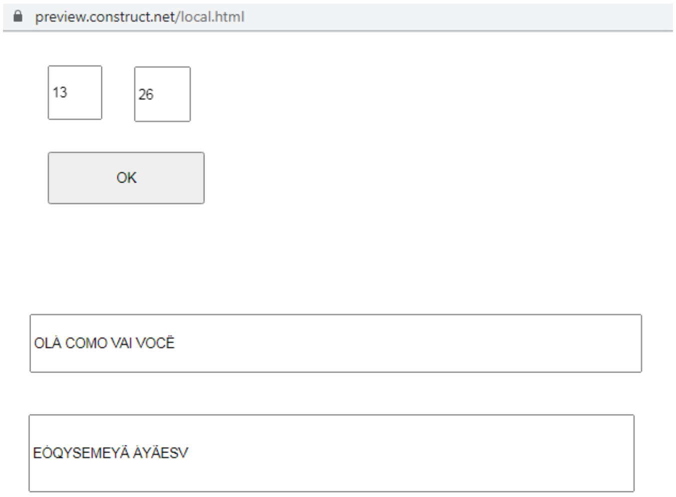

Figura 3: Programa para criptografar em funcionamento..

Neste caso, precisaremos de três *Entradas de texto* com o *Tipo* definido como *Número*, nomeadas como *M*, *A* e *B*, um *Botão*, e dois objetos do tipo *Texto*, nomeados como *Texto* e *Texto2* (Fig. 4).

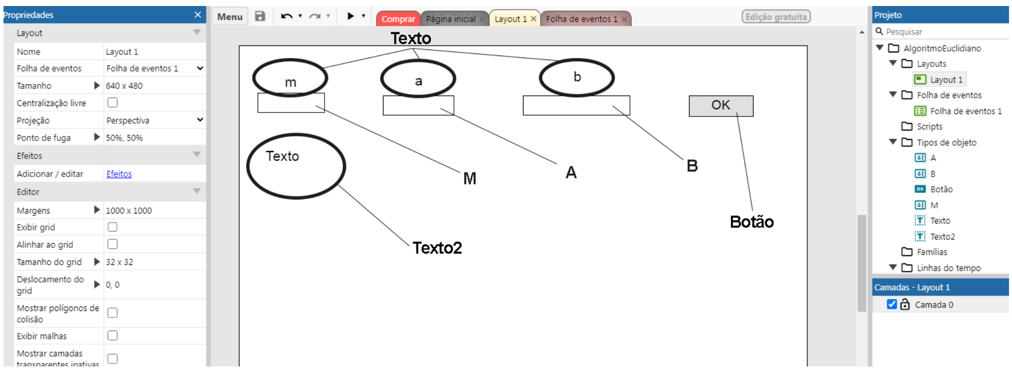

Figura 4: Interface gráfica do programa do Algoritmo Euclidiano.

Vamos adaptar e implementar o Algoritmo Euclidiano estendido (WIKIPEDIA, 2022) descrito logo abaixo. Basicamente, este algoritmo faz o mesmo procedimento exibido nos Exemplos 5 e 6.

```
mdc_estendido(a, m)
(r_antigo, r) := (m, a)
(t_antigo, t) := (0, 1)
Enquanto r > 0 faça
     q := quociente da divisão de r_antigo por r
     (r_antigo, r) := (r, r_antigo - q * r)
     (t_antigo, t) := (t, t_antigo - q * t)
Saída "mdc(a,m) = ", r_antigo
Se r_antigo = 1, então escreva "Inverso multiplicativo de a em congruências módulo m :", t_antigo
Se r_antigo ≠ 1, então escreva "a não tem inverso multiplicativo em congruências módulo m."
```

Para o programa funcionar corretamente, lembramos que os valores inteiros de $a > 0$ e $b \geq 0$ devem ser menores que $m > 1$. A *Folha de eventos* da implementação deste Algoritmo Euclidiano está na Fig. 5.

Seguindo o Exemplo 5, caso seja digitado $a = 551$, $b = 0$ e $m = 874$, o programa retorna $\text{mdc}(551, 874) = 19$ e informa que 551 não possui inverso multiplicativo em congruência módulo 874.

Digitando $m = 34$, $a = 13$ e $b = 26$, o programa diz que $\text{mdc}(13, 34) = 1$ e informa os valores de $a' = 21$ e $b' = 32$, ou seja, a função $f(x) = 13x + 26$ tem função inversa $f^{-1}(x) = 21x + 32$ em congruências módulo 34. Neste caso, a chave de codificação $a = 13$ e $b = 26$, para um alfabeto de $m = 34$ caracteres, possui chave de decodificação $a' = 21$ e $b' = 32$. Perceba que os valores são os mesmo utilizados no Exemplo 6.

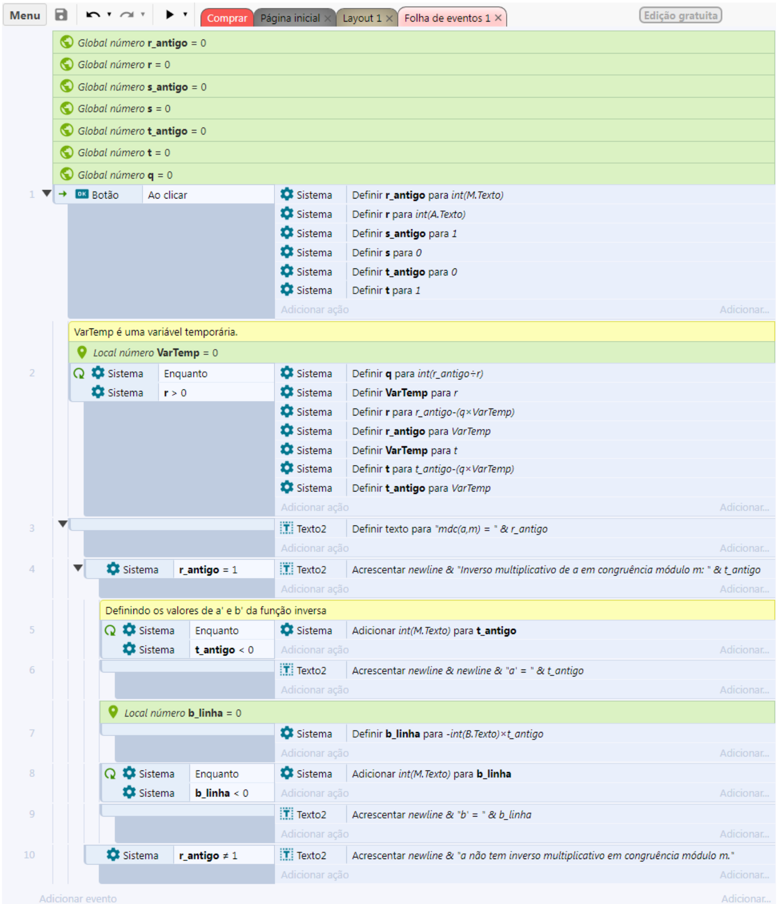

Figura 5: *Folha de eventos* da implementação do Algoritmo Euclidiano estendido.

4 Conclusão

Neste texto, apresentamos um método de criptografia utilizando uma função afim e como obter, através do Algoritmo Euclidiano, chaves de codificação e decodificação. Esperamos que os leitores se aprofundem mais no tema, melhorando os procedimentos aqui apresentados e pesquisando acerca de métodos mais elaborados, por exemplo: adicionando mais caracteres ao alfabeto utilizado (MENEZES NETO, 2021), criptografando em blocos de mais de um caractere, e buscando outros métodos de criptografia, como é o caso do sistema criptográfico assimétrico RSA (GANASSOLI; SCHANKOSKI, 2015; LEMOS, 2010).

Financiamento

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

5 Bibliografia

EXTENDED euclidean algorithm. *In:* WIKIPEDIA: the free encyclopedia. [San Francisco, CA: Wikimedia Foundation], 2022. Disponível em: `https://en.wikipedia.org/w/index.php?title=Extended_Euclidean_algorithm&oldid=1089794322`. Acesso em: 01 set. 2022.

GANASSOLI, A. P.; SCHANKOSKI, F. R. **Criptografia e matemática.** Curitiba: [*S. n.*], 2015. Disponível em: `http://www.educadores.diaadia.pr.gov.br/arquivos/File/fevereiro2016/matematica_dissertacoes/dissertacao_fernanda_ricardo_scankoski.pdf`. Acesso em: 01 out. 2022.

HEFEZ, A. **Aritmética**. 2. ed. Rio de Janeiro: SBM, 2016.

LEMOS, M. **Criptografia, números primos e algoritmos.** 4. ed. Rio de Janeiro: IMPA, 2010. Disponível em: `https://impa.br/wp-content/uploads/2017/04/PM_04.pdf`. Acesso em: 09 set. 2022.

MENEZES NETO, J. L. **Primeiros passos em criptografia.** João Pessoa: Editora UFPB, 2021. Disponível em: `http://www.editora.ufpb.br/sistema/press5/index.php/UFPB/catalog/book/540`. Acesso em: 01 set. 2022.

MIRANDA, A. A. N. de S.; PAULA, F. V. de. Uma proposta para o ensino de funções afins por meio da criptografia. **REAMEC** - Rede Amazônica de Educação em Ciências e Matemática, v. 9, n. 2, e21059, 2021. Disponível em: `https://doi.org/10.26571/reamec.v9i2.12652`. Acesso em: 01 out. 2022.