

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

**CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE
TECNOLOGIA**

CHRISTOPHER OLIVEIRA	RA:18726430
GIULIANO SANFINS	RA:17142837
MATHEUS MORETTI	RA:18082974
MURILO ARAUJO	RA:17747775
VICTOR REIS	RA:18726471

SISTEMAS OPERACIONAIS B – PROJETO 1

CRIPTO – DEVICE – DRIVER – SOB

**CAMPINAS
2020**

SUMÁRIO

1. INTRODUÇÃO	3
2. DETALHES DE PROJETO DO MÓDULO DE KERNEL	3
3. DESCRIÇÃO DOS RESULTADOS	3
4. CONCLUSÃO	21

1. INTRODUÇÃO

Esse projeto teve como principal objetivo familiarizar o aluno com os principais aspectos de implementação de um módulo de kernel, desde a criação, compilação, instalação e testes, sendo esse módulo é capaz de utilizar a API criptográfica do kernel Linux, para cifrar, decifrar e calcular o *hash*.

Um outro objetivo seria analisar e entender o funcionamento de algoritmos que trabalham com e sem criptografia no *Linux*, analisar seus resultados e além disso verificar os códigos que trabalham com cálculo de *hash*, assim gerando uma base no conhecimento de noções de segurança de dados.

2. DETALHES DE PROJETO DO MÓDULO DE KERNEL

Neste projeto alguns objetivos foram requisitados, dois códigos precisavam ser escritos, um sendo o módulo que faria, as chamadas para a API criptográfica do kernel, a conversão da *string* inserida para hexadecimal e a conversão oposta, transformando a *string* hexadecimal de volta para a *string* original, além de realizar o resumo criptográfico (*HASH*), e outro código que trabalharia em nível de usuário onde receberia as requisições e *strings* enviadas pelo o usuário.

3. DESCRIÇÃO DOS RESULTADOS

Para entendermos melhor o funcionamento da criptografia, foi nos dado um trabalho onde deveríamos inserir uma frase (*string*), e após isso convertê-la em hexadecimal e analisar o que ocorre com a *string* utilizando e não usando criptografia, para saber a diferença do que ocorre quando é aplicada ou não, podendo observar a partir desse simples teste as metodologias para se obter a *string* original. Após realizarmos a construção da criptografia e toda essa análise através de impressões no terminal do Linux, foi desenvolvido uma segunda função, no qual tinha principal intuito de “*descriptografar*” a frase inserida pelo usuário, e também, realizar uma segunda conversão, que deveria obter a frase que está convertida em hexadecimal, passando-a para a frase original que foi escrita pelo usuário. Essa frase quando continha uma quantidade menor de 16 bytes, era completada com zeros em suas últimas posições após a entrada do usuário.

No começo tivemos algumas dificuldades, pois não saberíamos como realizar essa transformação e quantos “bytes” e posições em um vetor cada letra ou número ocuparia. Então após a realização de alguns testes em um arquivo separado, e

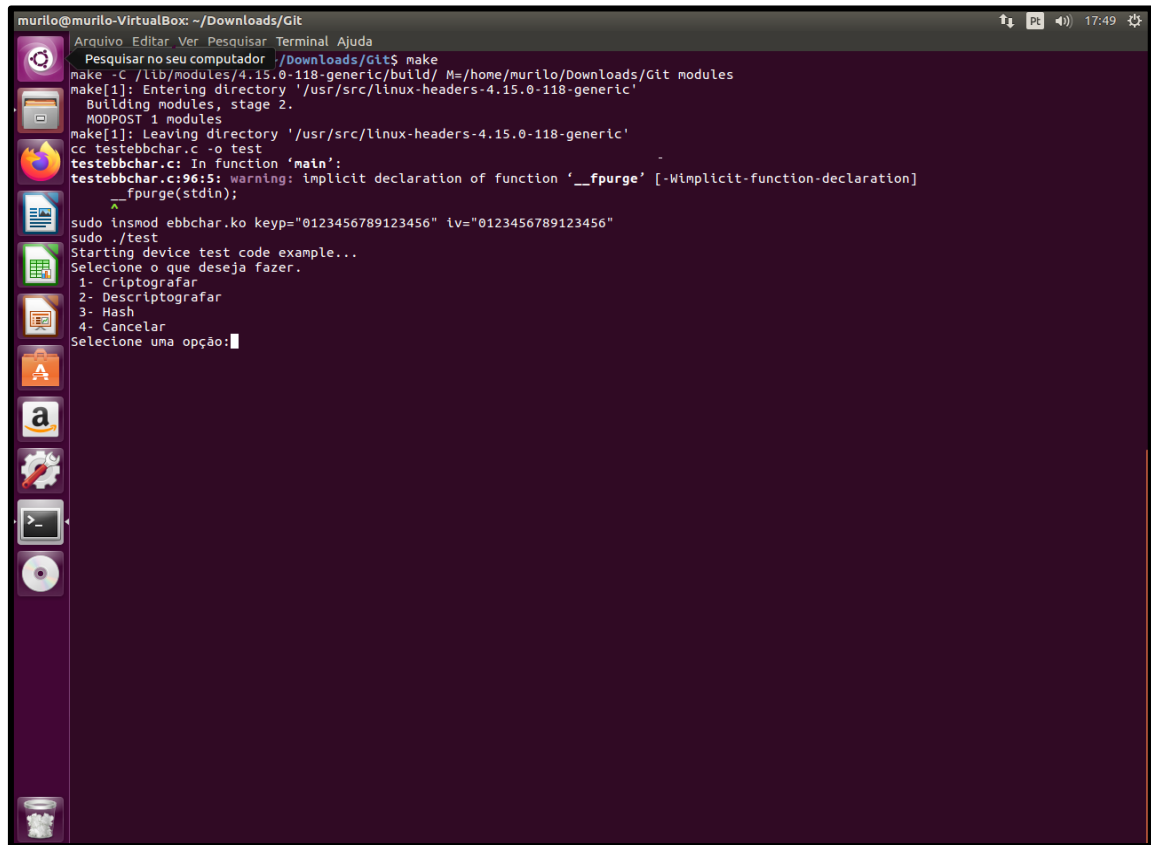
analisado como seria a melhor forma de aplicar essas duas conversões, podendo notar que uma letra ou um número poderia ser escrito em hexadecimal com duas posições em um vetor, sendo a letra acompanhada de um número ou até mesmo de outra letra na segunda posição do vetor, onde através disso foi realizado a manipulação de dividir por 16 e trabalhar com o resto da divisão, e depois, o processo inverso, de multiplicar por 16 elevado a 0 ou 1, pois se tratava apenas de duas posições. Podendo dizer que após vários testes e análise foi possível obter a frase de origem exatamente como foi escrita, aplicando depois a criptografia nessa frase como foi dito acima o seu objetivo. Nessa criptografia, tinha que conter sempre uma chave e um *iv*.

Por fim, foi desenvolvido a função responsável pelo cálculo do *hash* para trabalharmos melhor com os dados e análise de seus resultados, no qual, especificamente no *hash*, não completamos a *string* inserida pelo usuário com zeros.

Abaixo foi inserido uma sequência de imagens em um dos testes finais que realizamos do programa, onde é possível verificar o menu e as opções de criptografia (c), *descriptografia* (d) e *hash* (h). Da figura 1 a figura 5, pode observar a escolha de criptografia no menu, e a inserção de entrada de *string*, onde foi digitado “*abcde12345*”, e depois convertendo esse valor em hexadecimal, onde foi retornado “*61626364653132333435303030303030*” como pode ser visto na figura 4, e depois aplicando a criptografia, que foi utilizado o *keyp* de “*0123456789123456*” e *iv* de “*0123456789123456*”, retornando o resultado “*69e62d6aB9847e7f9cb9864101406dbd*”, copiando esse valor e aplicando na opção de “*descriptografia*”, como pode ver com mais detalhe da figura 6 a 10, observando que resulta no valor “*61626364653132333435303030303030*”, pois após aplicar a função de “*descriptografia*” e “*desconversão*”, resultado na *string* em hexadecimal criptografada na opção 1.

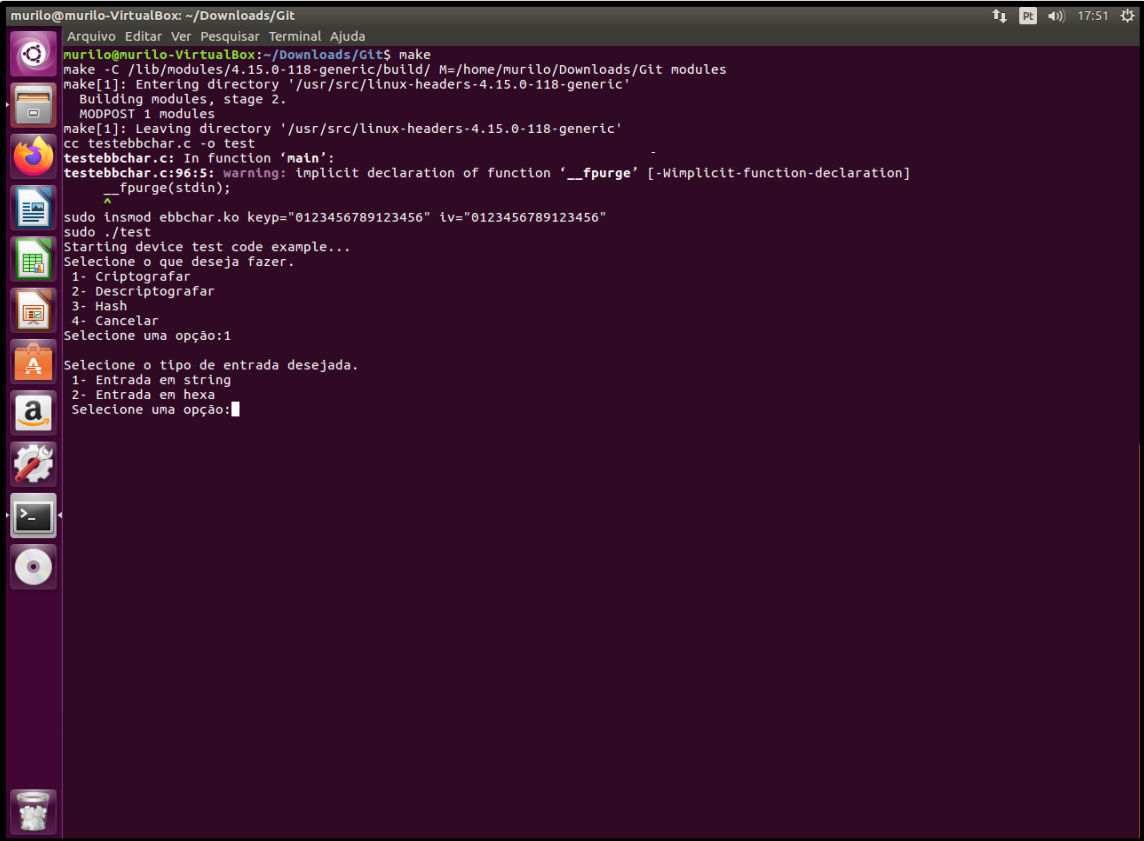
E também, as figuras 11 a 12, pode se ver com mais detalhe o funcionamento da opção *hash*, onde foi inserido também a frase “*abcde12345*”, e retornado o valor “*61d6504733ca7757e259c644acd085c4dd471019*”, onde foi realizado com a chave *SHA1* (*CryptoLogic SHA1 valor; 40 caracteres*) para o cálculo do *hash*, e por fim, na figura 13, com um auxílio de uma calculadora de valor de *hash* online, foi possível confirmar que o nosso cálculo do *hash* funciona corretamente como o esperado.

E por fim, nas figuras 14 e 15, mostra a interação de saída de do programa, mostrando como funciona se caso o usuário desejar encerrar o processo e como o nosso programa se comporta e imprime na tela a saída.



```
murilo@murilo-VirtualBox: ~/Downloads/Git
Arquivo Editar Ver Pesquisar Terminal Ajuda
Pesquisar no seu computador ./Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function 'f__fpurge' [-Wimplicit-function-declaration]
    f__fpurge(stdin);
    ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção: 1
```

Figura 1: Tela Inicial onde o usuário seleciona o que deseja fazer, criptografar, *descriptografar* ou calcular o resumo criptográfico de uma futura entrada, ou cancelar para fechar o programa.

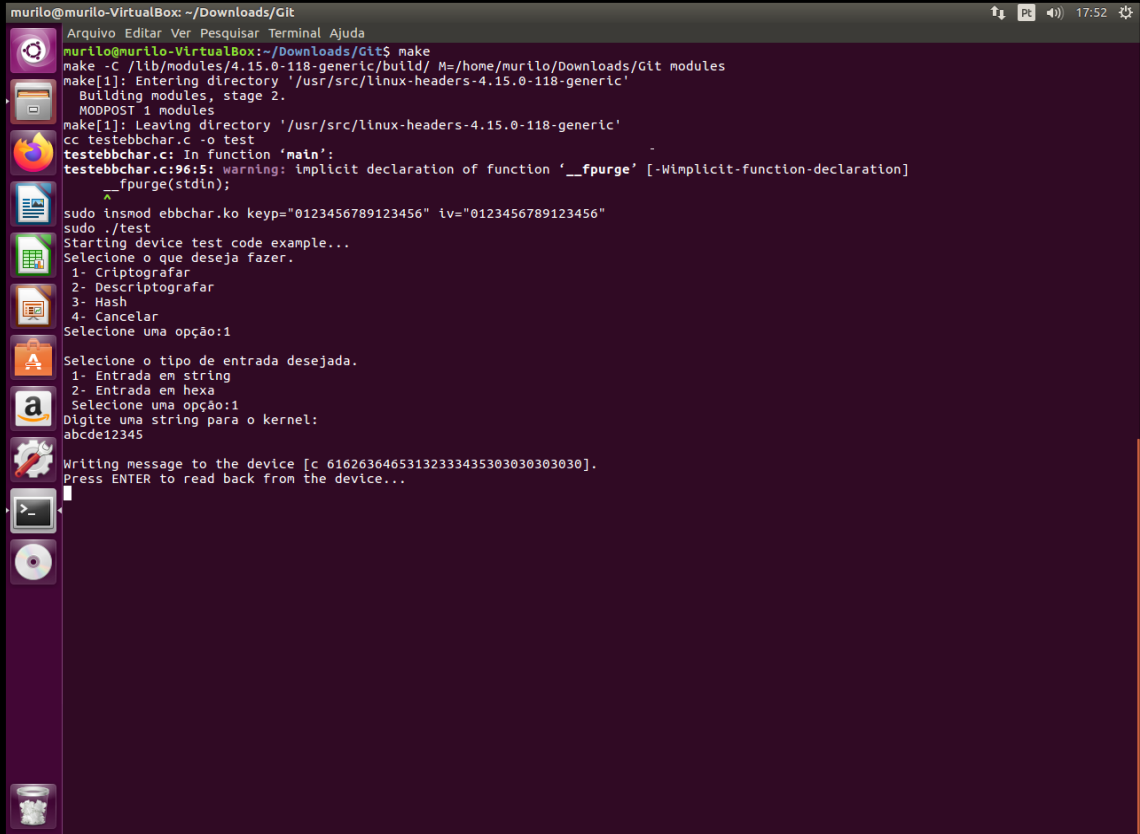


```
murilo@murilo-VirtualBox: ~/Downloads/Git
Arquivo Editar Ver Pesquisar Terminal Ajuda
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
     __fpurge(stdin);
     ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:1
Selecione o tipo de entrada desejada.
1- Entrada em string
2- Entrada em hexa
Selecione uma opção:1
```

Figura 2: Tela secundária onde o usuário seleciona o tipo de entrada que colocará, podendo ser *string* comum ou uma *string* hexadecimal.

```
murilo@murilo-VirtualBox: ~/Downloads/Git
Arquivo Editar Ver Pesquisar Terminal Ajuda
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
     __fpurge(stdin);
     ^
^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:1
Selecione o tipo de entrada desejada.
1- Entrada em string
2- Entrada em hexa
Selecione uma opção:1
Digite uma string para o kernel:
```

Figura 3: Neste exemplo foi selecionado a função de criptografia utilizando como entrada uma *string* comum.



```
murilo@murilo-VirtualBox: ~/Downloads/Git
Arquivo Editar Ver Pesquisar Terminal Ajuda
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
ce testebbchar.c -o test
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
    __fpurge(stdin);
    ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:1
Selecione o tipo de entrada desejada.
1- Entrada em string
2- Entrada em hexa
Selecione uma opção:1
Digite uma string para o kernel:
abcde12345
Writing message to the device [c 61626364653132333435303030303030].
Press ENTER to read back from the device...
```

Figura 4: Ainda com a configuração anterior, foi redigido a entrada “abcde12345”, e o primeira impressão se trata da entrada completada com 0’s (zeros) (para completar os bits faltantes), e então convertida em Hexadecimal.


```
murilo@murilo-VirtualBox: ~/Downloads/Git
Arquivo Editar Ver Pesquisar Terminal Ajuda
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
    __fpurge(stdin);
    ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descritografar
3- Hash
4- Cancelar
Selecione uma opção:1
Selecione o tipo de entrada desejada.
1- Entrada em string
2- Entrada em hexa
Selecione uma opção:1
Digite uma string para o kernel:
abcde12345
Writing message to the device [c 61626364653132333435303030303030].
Press ENTER to read back from the device...
Reading from the device...
The received message is: [69e62d6a89847e7f9cb9864101406dbd]
Press ENTER to clean the screen...
```

Figura 5: Continuando o exemplo anterior, a segunda impressão se trata da mensagem criptografada utilizando a API criptográfica do Kernel.

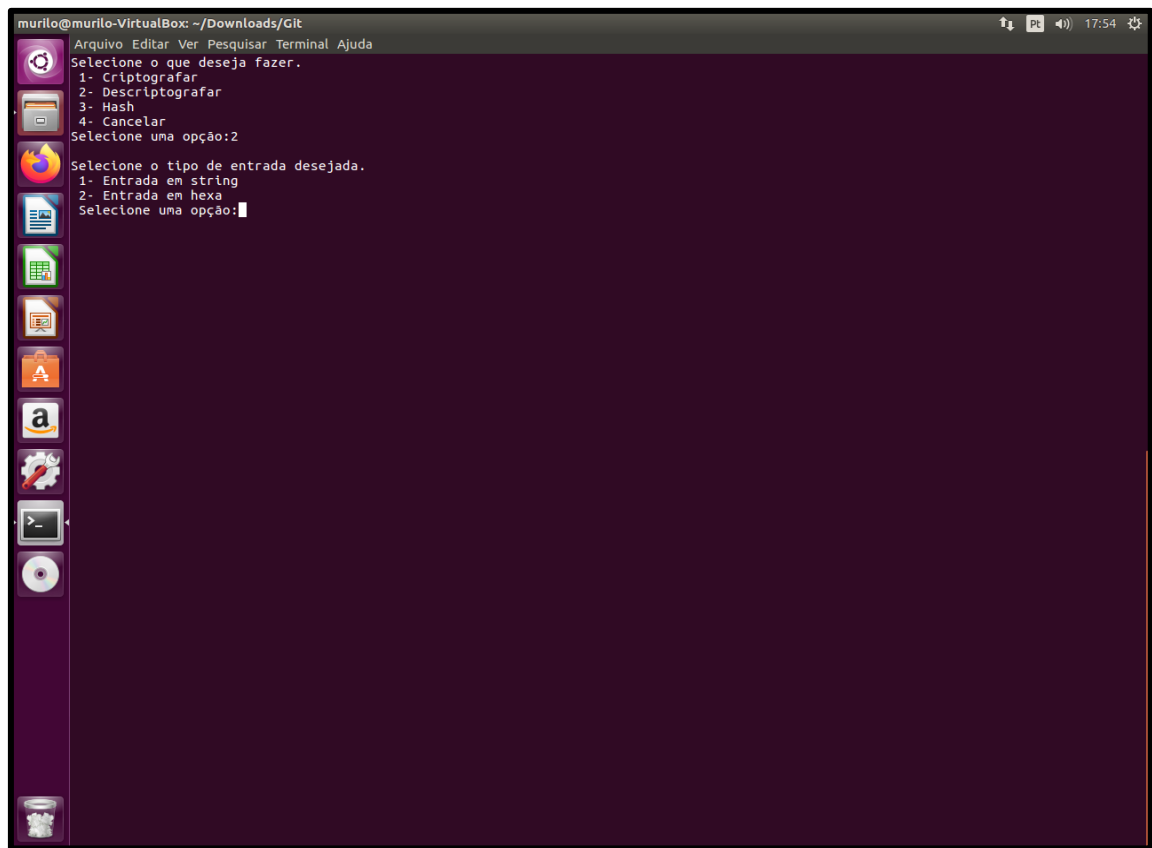


Figura 6: Neste exemplo foi selecionado a função de *descriptografia*.

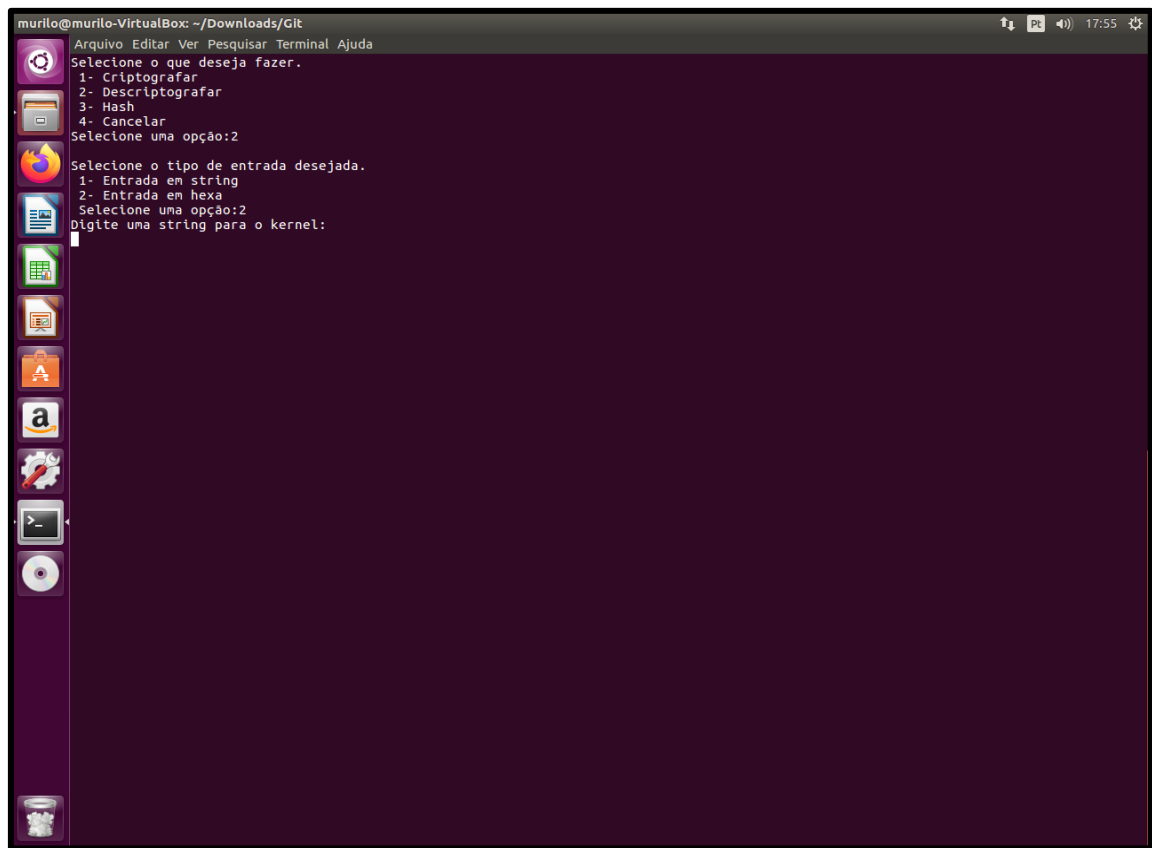


Figura 7: E a entrada selecionada foi uma *string* Hexadecimal.

```
murilo@murilo-VirtualBox: ~/Downloads/Git
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
    __fpurge(stdin);
    ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:2
Selecione o tipo de entrada desejada.
1- Entrada em string
2- Entrada em hexa
Selecione uma opção:2
Digite uma string para o kernel:
69e62d6a89847e7f9cb9864101406dbd
```

Figura 8: Sendo a mensagem redigida “69e62d6a89847e7f9cb9864101406dbd”.

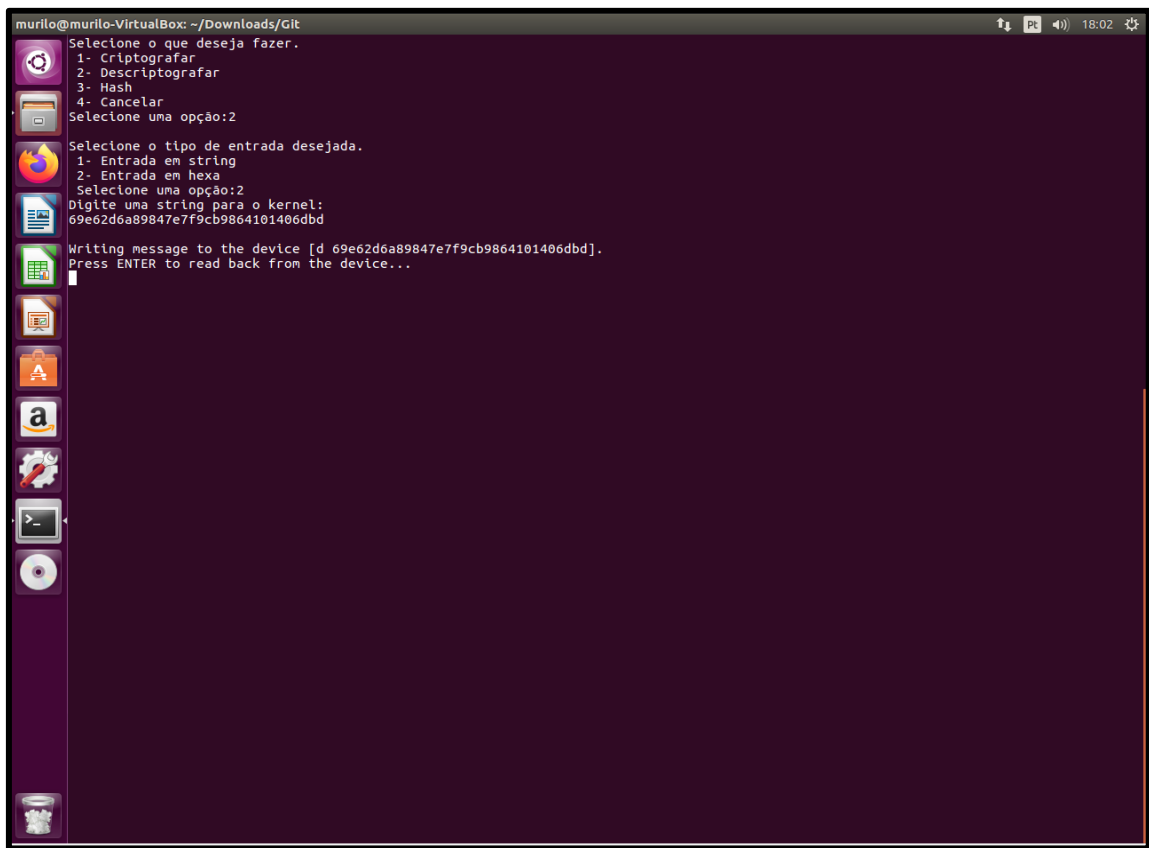


Figura 9: A impressão apresentada se trata da própria mensagem enviada por se tratar de ser uma *string* Hexadecimal.

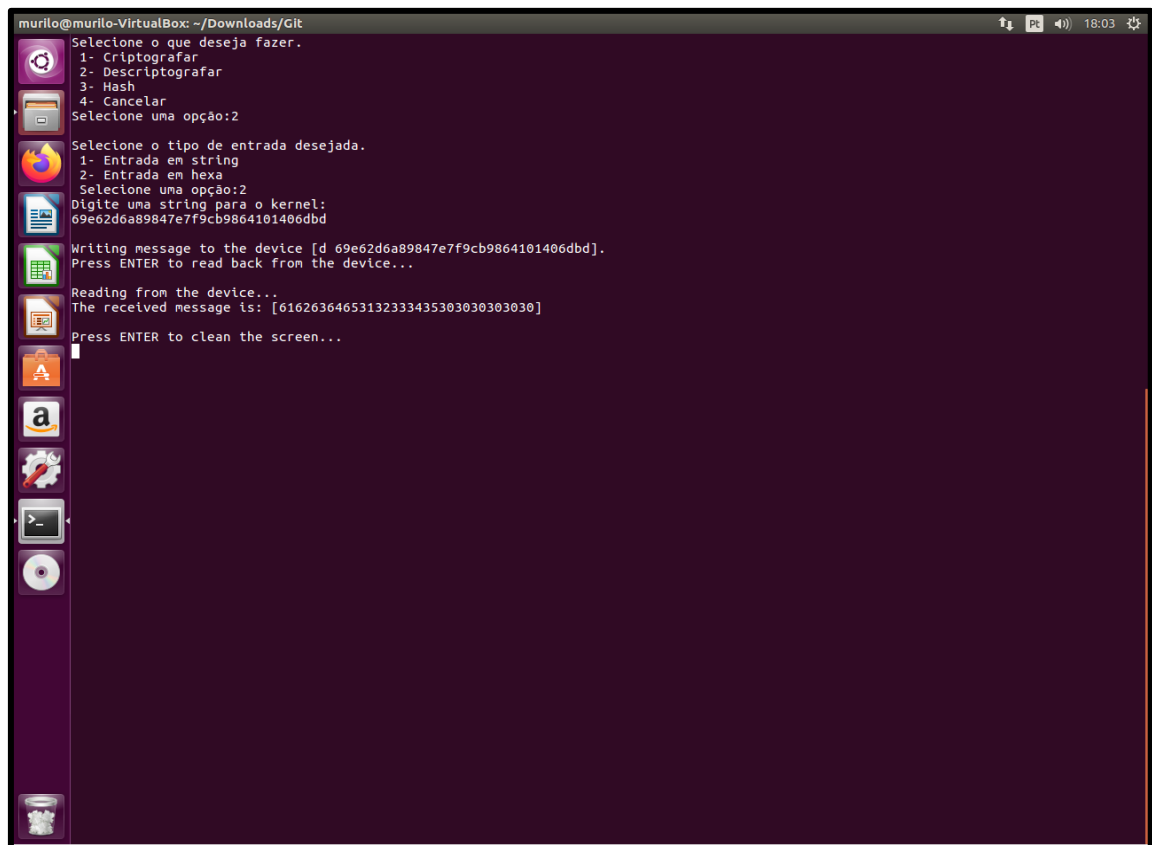


Figura 10: E a segunda impressão trata-se da mensagem enviada *descriptografada* utilizando a API criptográfica do Kernel, é possível identificar que a mensagem criptografada e *descriptografada* são as mesmas, como na figura 5.

```
murilo@murilo-VirtualBox: ~/Downloads/Git
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
     __fpurge(stdin);
     ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
[sudo] senha para murilo:
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:3
Digite uma string para o kernel:
abcde12345
```

Figura 11: Neste passo foi escolhido a opção de *hash* (resumo criptográfico) e como entrada foi inserido “abcde12345”.

```
murilo@murilo-VirtualBox: ~/Downloads/Git
murilo@murilo-VirtualBox:~/Downloads/Git$ make
make -C /lib/modules/4.15.0-118-generic/build/ M=/home/murilo/Downloads/Git modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-118-generic'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-118-generic'
cc testebbchar.c -o test
testebbchar.c: In function 'main':
testebbchar.c:96:5: warning: implicit declaration of function '__fpurge' [-Wimplicit-function-declaration]
    __fpurge(stdin);
    ^
sudo insmod ebbchar.ko keyp="0123456789123456" iv="0123456789123456"
[sudo] senha para murilo:
sudo ./test
Starting device test code example...
Selecione o que deseja fazer.
1- Criptografar
2- Descriptografar
3- Hash
4- Cancelar
Selecione uma opção:3
Digite uma string para o kernel:
abcde12345
Writing message to the device [h 61626364653132333435].
Press ENTER to read back from the device...
Reading from the device...
The received message is: [61d6504733ca7757e259c644acd085c4dd471019]
Press ENTER to clean the screen...
```

Figura 12: Para o *Hash* não é utilizada a conversão hexadecimal mesmo ela aparecendo na tela. O resultado do resumo é mostrado na tela na segunda saída.

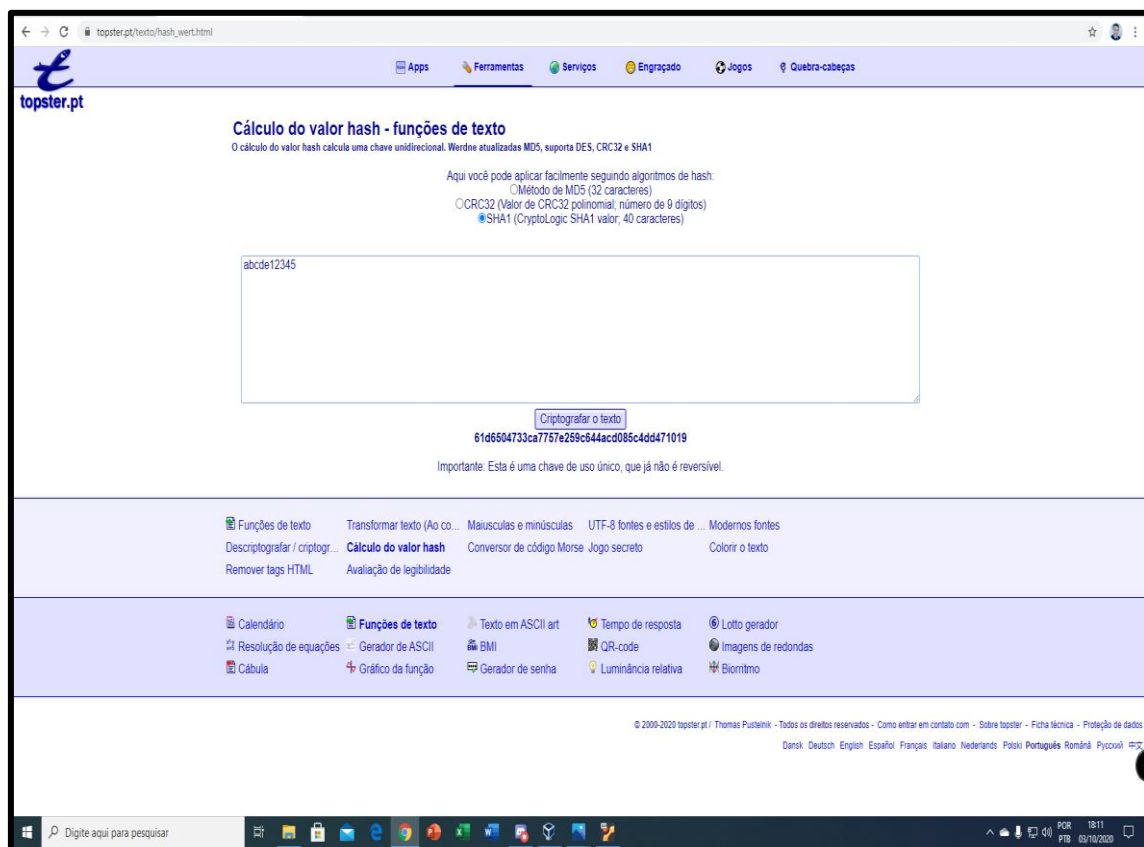


Figura 13: Foi feita uma comparação do resumo obtido em nosso módulo com o de um site, certificando-nos do mesmo estar correto.

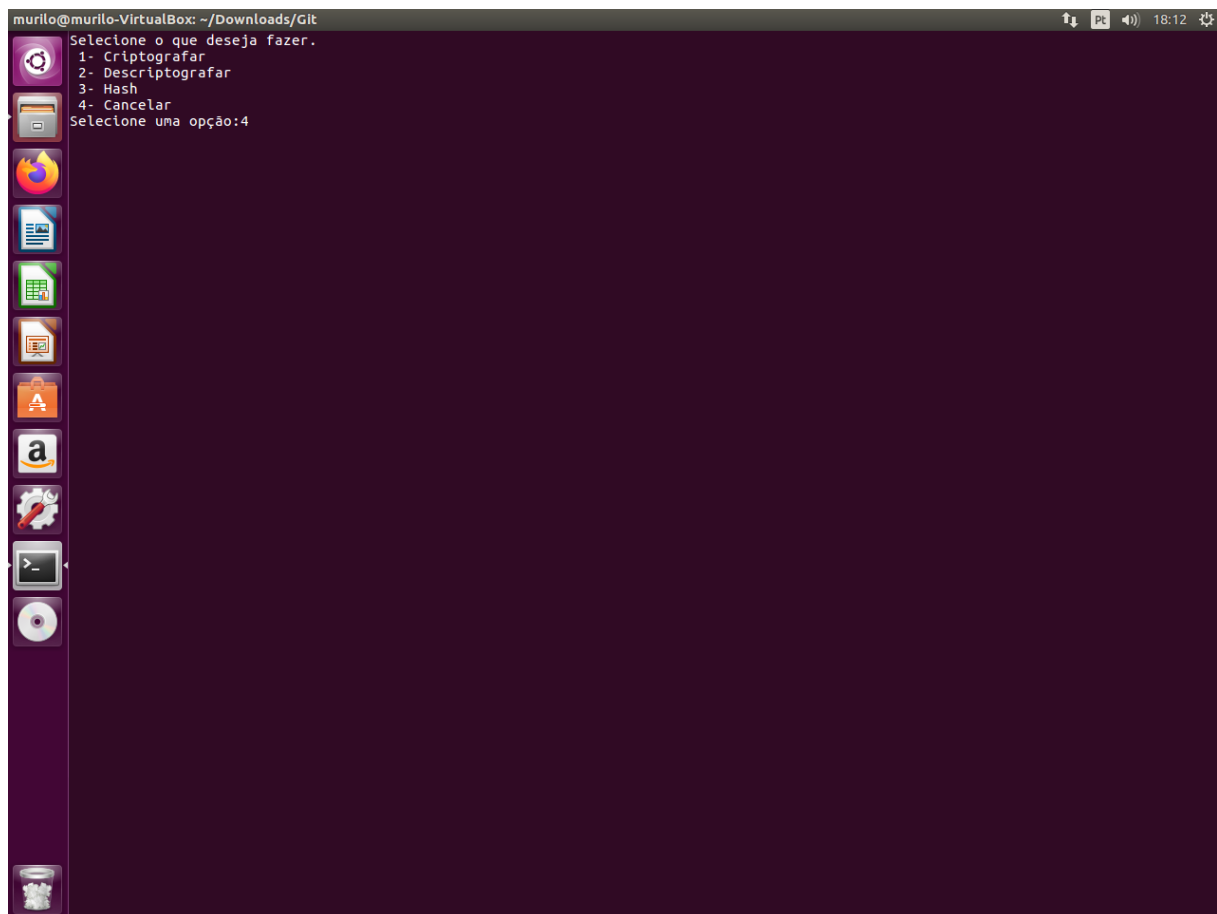


Figura 14: E por fim pode-se selecionar finalizar o programa de teste do módulo.

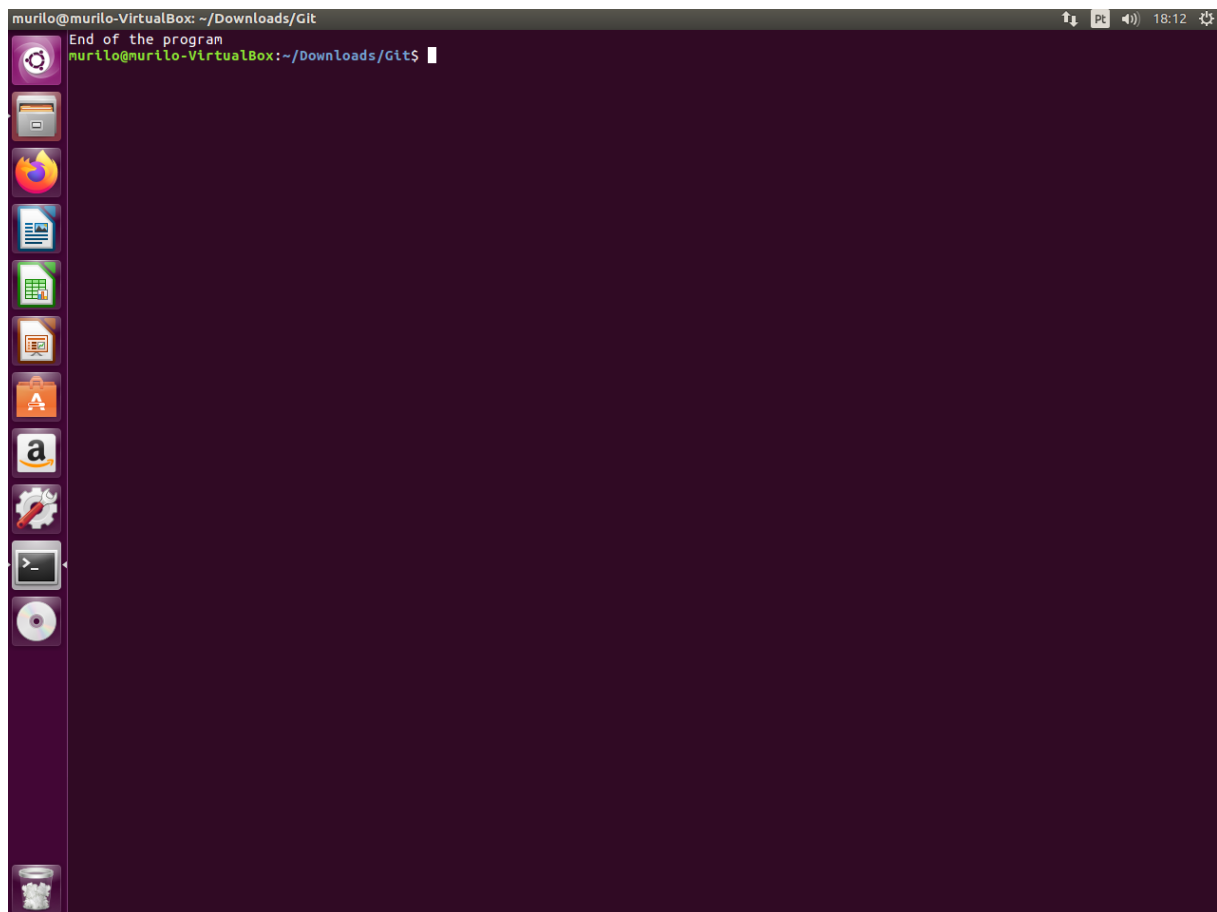
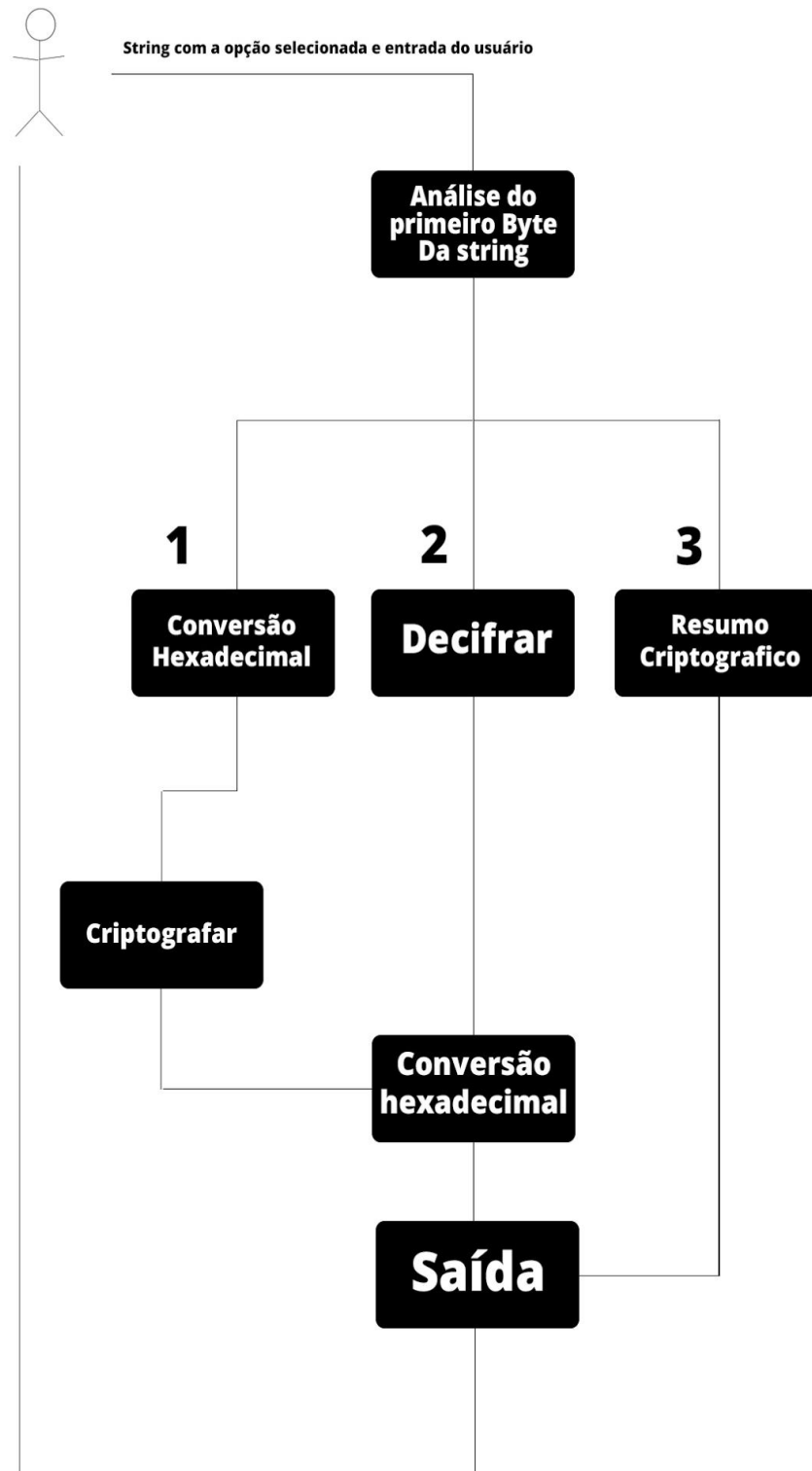


Figura 15: O programa é encerrado e a tela é limpa.

Abaixo pode se observar com mais detalhe um diagrama no qual foi construído, para melhor visualização de como o nosso programa funciona e se comporta em relação as devidas interações do usuário que estão sendo indicados nos retângulos do diagrama:



4. CONCLUSÃO

De acordo com os resultados obtidos é possível concluir que o objetivo principal do projeto foi alcançado com êxito, todos os aspectos da implementação do módulo, que utiliza a API criptografia do Kernel, foram realizados, testados e analisando o comportamento do algoritmo sem criptografia e com criptografia, verificando o que ocorre na *string* quando ela é codificada em hexadecimal, e também, de hexadecimal para o texto original.

Com esse experimento pudemos conhecer mais sobre o *Linux* no que se diz respeito aos módulos de kernel e como trabalhar com a API criptografia, como criar, compilar, instalar e remover um módulo, habilidades que serão requisitadas no decorrer dos próximos projetos e até mesmo no ramo empresarial e profissional se um dia for requisitado.