

Pontifícia Universidade Católica de Campinas Faculdade de Engenharia de Computação - FECOMP

Sistemas Operacionais B – Relatório Experimento 1

Módulo de criptografia do kernel Linux

Beatriz Morelatto Lorente RA: 18071597

Cesar Marrote Manzano RA: 18051755

Fabricio Silva Cardoso RA: 18023481

Pedro Ignácio Trevisan RA: 18016568

**Sumário**

**1. Introdução.......................................................................................................3**

**2. Algoritmos de criptografia utilizados...........................................................4**

**3. Programa de teste..........................................................................................5**

**4. Módulo de criptografia...................................................................................6**

**5.Testes e resultados.......................................................................................11**

**6. Conclusão.....................................................................................................16**

**Introdução**

O experimento desenvolvido pretende demonstrar os passos feitos para o desenvolvimento de um módulo de criptografia do kernel Linux. O módulo é responsável por cifrar, decifrar ou calcular o resumo criptográfico de uma string fornecida pelo usuário. Para cifrar e decifrar uma string foi utilizado o algoritmo AES em modo CBC e para o resumo criptográfico o algoritmo SHA1. Para o teste do módulo foi criado um programa em espaço do usuário que se conecta com o dispositivo, enviando as requisições necessárias e exibindo o resultado da criptografia.

**Algoritmos de criptografia utilizados**

Para a cifrar e decifrar a string fornecida pelo usuário foi utilizado o algoritmo AES em modo CBC.

Primeiramente é necessário entender como o algoritmo AES (Advanced Encryption Standard) funciona. O algoritmo consiste em uma criptografia simétrica de blocos de tamanho fixo (utilizamos blocos de 16 bytes, 128 bits). O algoritmo usa uma chave para cifrar e decifrar os blocos e esta também tem 16 bytes.

No modo de criptografia CBC, para cada bloco de texto é aplicado a função XOR com o bloco anterior, garantindo que cada bloco seja dependente um do outro. Para o primeiro bloco, é aplicado um vetor de inicialização, garantindo que cada operação de criptografia seja única. O esquema do algoritmo é representado abaixo:

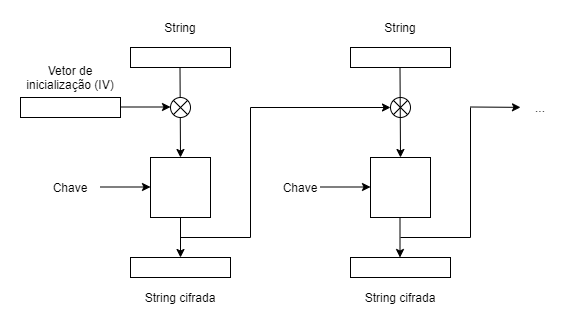


Figura 1 - Esquema do modo de criptografia CBC

Já para o resumo criptográfico foi utilizado o algoritmo SHA1. Consiste em uma função de dispersão criptográfica, considerado bem seguro. O algoritmo produz uma mensagem de dispersão com 20 bytes, que é conhecida como o resumo da mensagem original.

**Programa de teste**

Para testarmos o experimento foi feito um programa em espaço de usuário que se comunica-se com o módulo. O esquema abaixo representa como essa comunicação é feita.

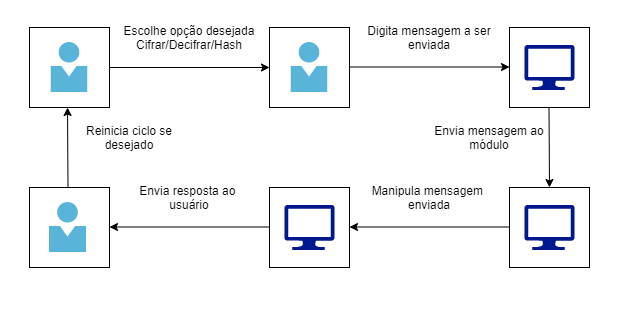


Figura 2 - Esquema do funcionamento do programa de teste

Primeiramente é escolhida a operação que se deseja fazer, ou seja, cifrar, decifrar ou calcular o resumo criptográfico de uma mensagem. Após a escolha o usuário digita qual a mensagem que será enviada ao módulo. Para saber qual operação será feita, foi concatenado as letras ‘c’ (cifrar), ‘d’ (decifrar) ou ‘h’ (hash), no final da mensagem, dependendo da escolha do usuário. Desse modo o módulo pode aplicar adequadamente o algoritmo em cima de uma mensagem, basta ler apenas a última posição da string para isso. Após manipular a mensagem (mais detalhes sobre essa manipulação serão discutidas mais para frente), o módulo devolve a mesma para o programa de teste e este se encarrega de imprimir as informações necessárias. Todo o ciclo pode ser feito novamente caso o usuário deseje.

**Módulo de criptografia**

O funcionamento do módulo de criptografia é bastante simples, ele apenas manipula uma mensagem e envia para o programa de teste. Porém, a manipulação da mensagem não é um processo fácil e diversos erros podem ser cometidos na hora de sua implementação. O esquema a seguir mostra como o módulo funciona.

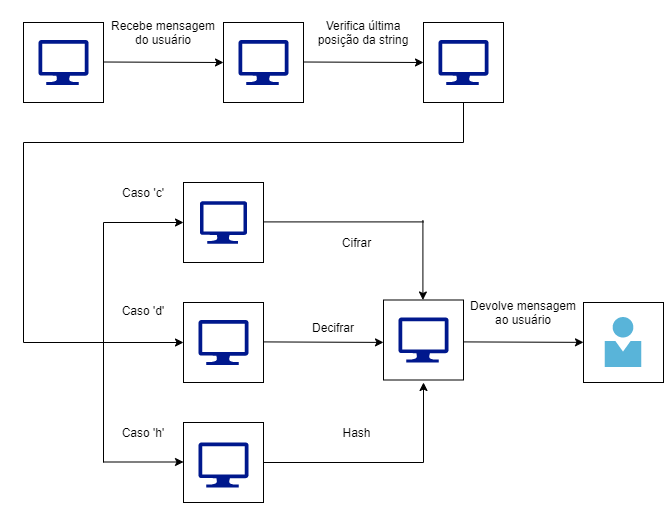


Figura 3 - Esquema do funcionamento do módulo de criptografia

Antes de prosseguir com a explicação sobre os algoritmos, é importante destacar alguns detalhes da comunicação entre os programas.

O uso de mutex para a comunicação

Para que não houvesse nenhum problema de sincronização, foi necessário o uso de mutex. Com esse detalhe, foi possível bloquear um processo em espaço de usuário, evitando que mais de um processo fizesse uma requisição para o módulo de criptografia. O mutex é bloquado quando o programa de teste tenta acessar (ler) o módulo e é liberado quando este fecha, ou seja, devolve uma mensagem. Para mais informações sobre o uso de mutex e a comunicação entre os programas, basta acessar o link a seguir: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device>. O esquema abaixo exemplifica o uso do mutex em nosso programa.

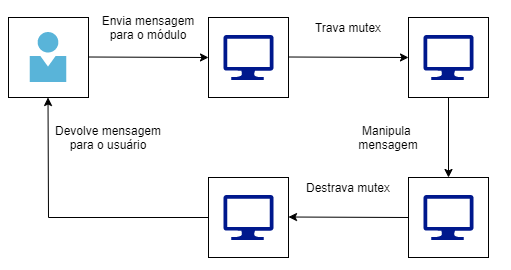


Figura 4 - Esquema do funcionamento do mutex

Detalhes de implementação das operações de cifrar e decifrar

Como citado anteriormente, o programa é capaz de fazer operações de cifrar e decifrar uma mensagem. O link a seguir, da documentação da API do kernel linux, contém explicações e exemplos de código de criptografia: <https://www.kernel.org/doc/html/v4.12/crypto/index.html>. Alguns detalhes da implementação dos algoritmos serão discutidos abaixo.

Primeiramente, é importante lembrar que para essas operações, foi usado o algoritmo AES em modo CBC. Portanto é necessário se utilizar de uma chave e um vetor de inicialização. Os valores desses componentes foi passado como parâmetro ao módulo e fixado com ‘0123456789ABCDEF’ em ambos os casos.

O primeiro passo para se implementar a função para cifrar(encrypt) e decifrar(decrypt), é dizer para o módulo qual algoritmo será usado e setar a estrutura de dados que será usada na operação. A imagem a seguir mostra como isso é feito.

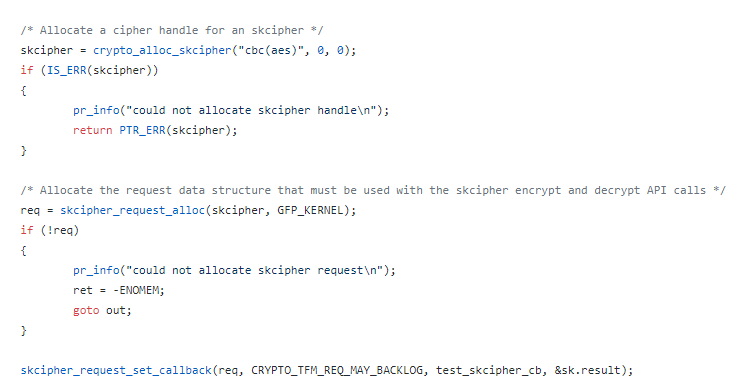


Figura 5 - Código da primeira parte de implementação para cifrar e decifrar

Com isso feito, é necessário requisitar um espaço em memória para alocar a chave e o vetor de inicialização. Como o tamanho de cada um é fixo em 16 bytes, basta alocar esse tamanho para ambos. A imagem a seguir mostra como isso é feito.

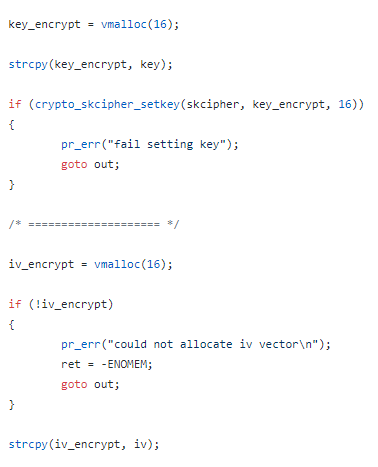


Figura 6 - Código da segunda parte de implementação para cifrar e decifrar

Também é necessário fazer o mesmo processo para a mensagem que é enviada, porém note que o espaço alocado para a mensagem não é fixo e sim correspondente ao tamanho da mesma.

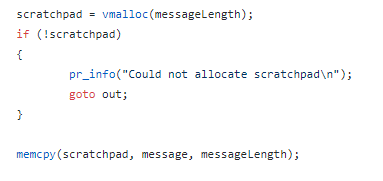


Figura 7 - Código da primeira parte de implementação para cifrar e decifrar

Por fim, basta setar os dados e chamar a função para cifrar ou decifrar a mensagem.

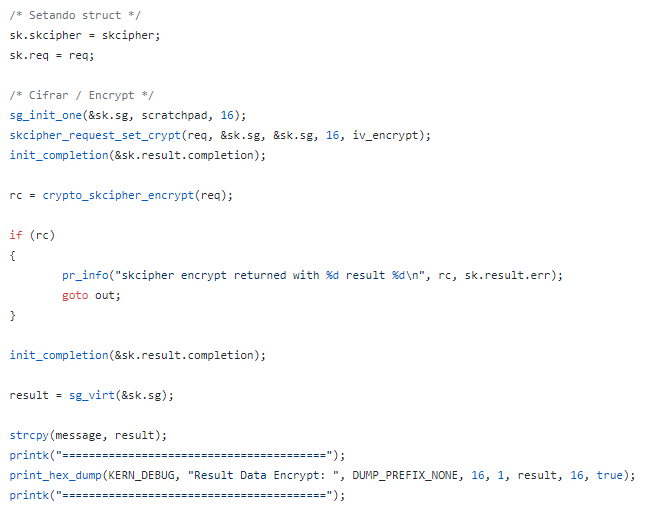


Figura 8 - Código da terceira parte de implementação para cifrar

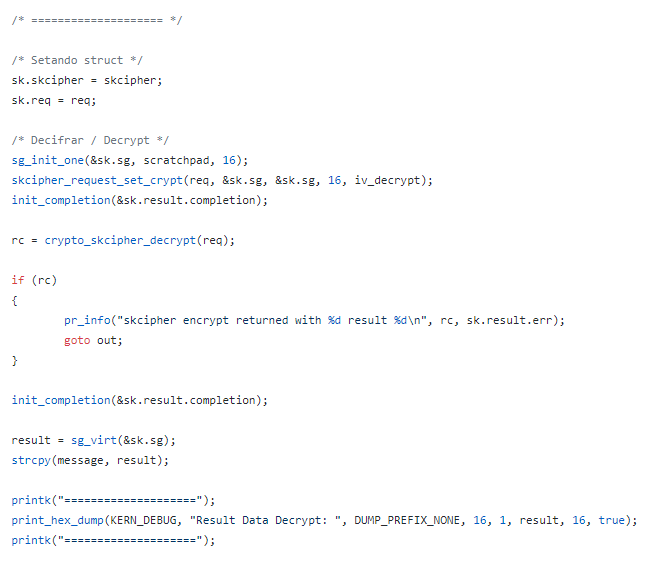


Figura 9 - Código da terceira parte de implementação para decifrar

Detalhes de implementação da operação de hash

Assim como as funções de cifrar e decifrar, a função de hash também é simples de implementar. Novamente utilizamos o link da documentação da API de criptografia do kernel Linux para nos auxiliar na implementação desta função: <https://www.kernel.org/doc/html/v4.12/crypto/index.html>.

O primeiro passo para a implementação é requisitar o algoritmo que será utilizado (no caso SHA1). Depois é necessário alocar um espaço em memória para alocar as estruturas que serão usadas e logo depois setar as mesmas. Também é necessário alocar um espaço para armazenar o resultado. Após esses passos, basta chamar a função de hash com os parâmetros necessários.



Figura 10 - Código da implementação da função hash

**Testes e Resultados**

Cifragem e Decifragem

Para os testes fotram usadas as seguintes strings:

* bom
* pedro
* cesar
* hello world
* hello world 2

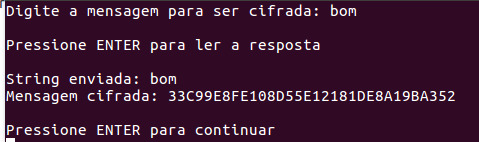


Figura 11 - Primeira string cifrada

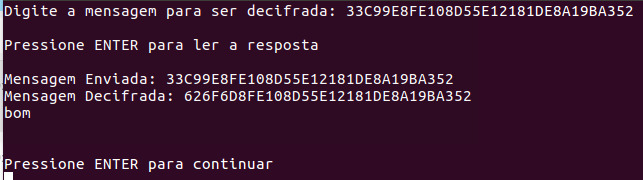


Figura 12 - Primeira string decifrada

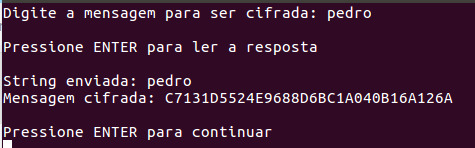


Figura 13 - Segunda string cifrada

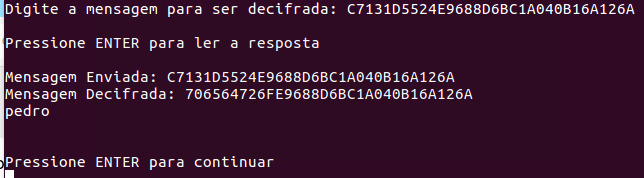


Figura 14 - Segunda string decifrada

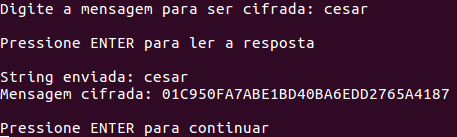


Figura 15 - Terceira string cifrada

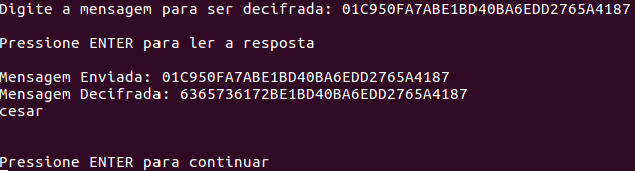


Figura 16 - Terceira string decifrada

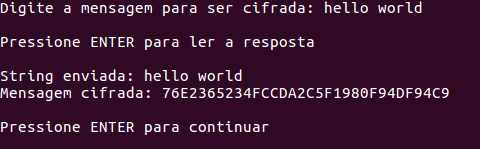


Figura 17 - Quarta string cifrada

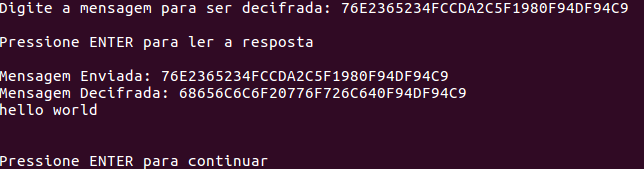


Figura 18 - Quarta string decifrada

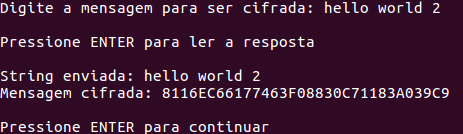


Figura 19 - Quinta string cifrada

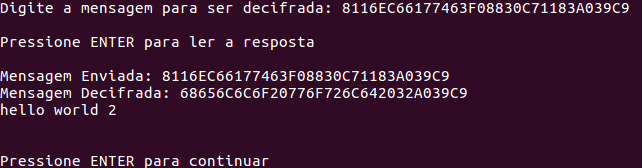


Figura 20 - Quinta string decifrada

Hash

Para os testes fotram usadas as seguintes strings:

* hello world
* testando o programa
* ordem e progresso
* um ninho de mafagafos tinha sete mafagafinhos

Para confirmar o resultado foi utilizado o site <http://www.sha1-online.com/>.

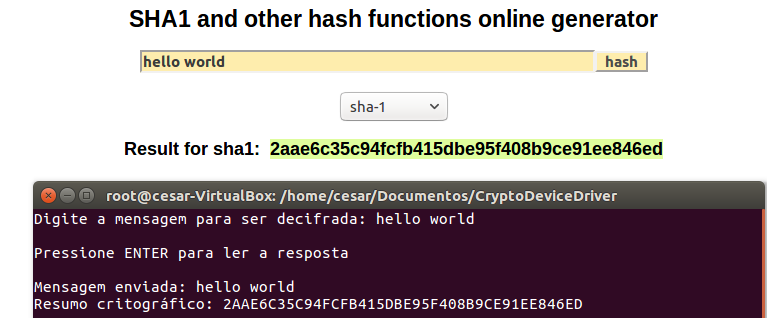


Figura 21 - Hash primeira string

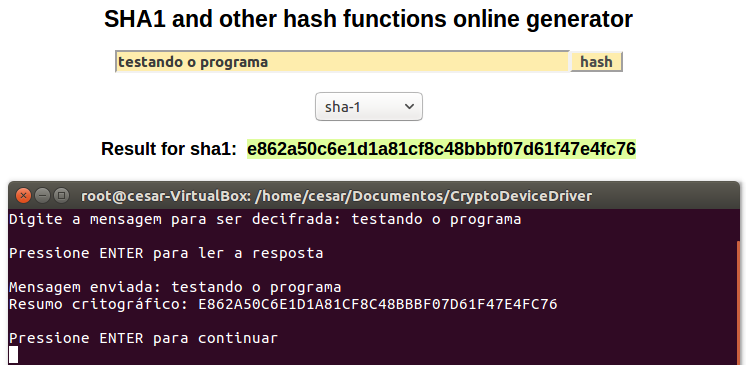


Figura 22 - Hash segunda string

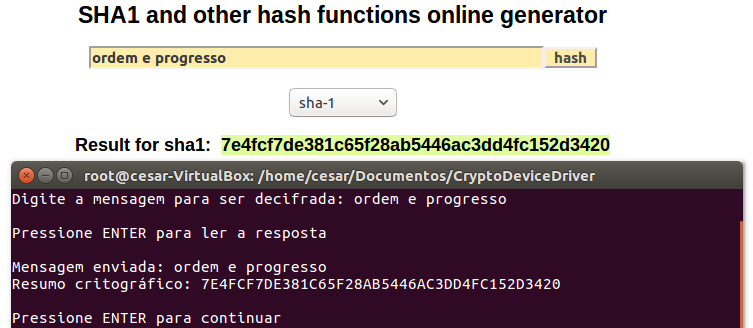


Figura 23 - Hash terceira string

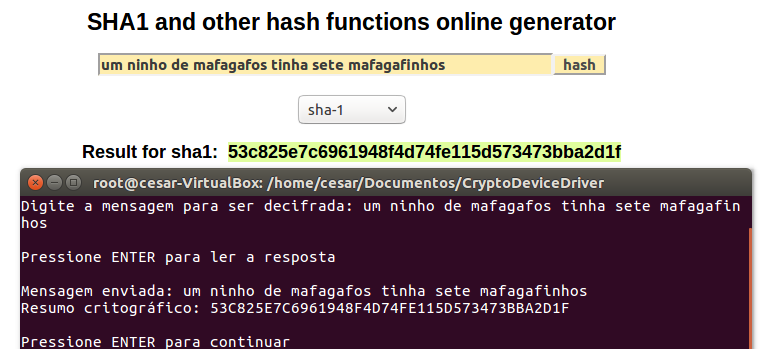


Figura 24 - Hash quarta string

**Conclusão**