

*Sistemas Operacionais B*

**Projeto 1  
*Crypto Device Driver***

Breno Baldovinotti | RA: 14315311

Caroline Gerbaudo Nakazato | RA: 17164260

Marco Antônio de Nadai Filho | RA: 16245961

Nícolas Leonardo Külzer Kupka | RA: 16104325

*24/10/2019*

*CEATEC – Centro de Ciências Exatas, Ambientais e de Tecnologia*

**1.Introdução**

A criptografia é de extrema importância no mundo atual, fornecendo segurança e confiabilidade às constantes trocas de informações. No Linux, não é diferente. Sua API criptográfica é amplamente utilizada em outras porções de kernel, e é fundamental para o correto funcionamento do sistema operacional, pois permitindo uma comunicação correta e segura com outros dispositivos ou até mesmo entre os próprios mecanismos do SO.

Este projeto visa a análise e o entendimento das técnicas de implementação de um módulo de kernel para o Linux que faz uso de sua API criptográfica presente no mesmo. Também, busca familiarizar-se com os detalhes de implementação de tal módulo de kernel e suas dependências.

Para isto, foi desenvolvido um módulo de kernel Linux versão de um dispositivo criptográfico, capaz de cifrar e decifrar mensagens através do algoritmo AES em modo CBC, utilizando-se de uma chave e um vetor de inicialização fornecidos na inserção do módulo. O dispositivo também deveria ser capaz de calcular o hash de uma string através do algoritmo SHA1.

Para o teste deste dispositivo, foi desenvolvido um programa em espaço de usuário que se comunica com o mesmo, podendo abrir o dispositivo, enviar uma requisição e exibir a resposta retornada ao usuário.

**2. Detalhes do Projeto**

O módulo foi desenvolvido na versão 4.1X do Kernel Linux, estando localizado no arquivo de dispositivo /dev/crypto. Seu Major Number é alocado dinamicamente. Quando o módulo do kernel for carregado, é necessário informar o parâmetro “key” (uma chave simétrica) e o parâmetro iv (vetor de inicialização), os quais são utilizados pelo algoritmo AES CBC para cifrar e decifrar os dados. Ambos os parâmetros correspondem a uma string representada em hexadecimal. O algoritmo de criptografia AES possuí um tamanho de bloco fixo em 128 bits, mas apresenta inúmeras variantes. Neste projeto, foi utilizada a variante AES CBC 128, o qual apresenta uma chave de comprimento de 128 bits.

O algoritmo do AES CBC, na operação de cifragem, utiliza-se da chave e do valor inicial (representados me hexadecimal) fornecidos durante a carga do módulo, retornando uma string cifrada correspondente aos dados fornecidos durante a requisição. A etapa de cifragem é realizada recursivamente com os novos dados gerados, até se obter a palavra cifrada final.

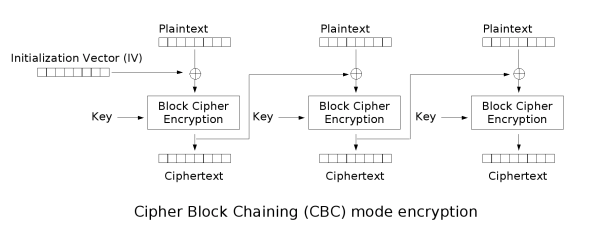


Figura 1: Diagrama do algoritmo de cifra AES em modo CBC.

Já na operação de decifragem, o processo inverso da cifragem é realizado, fornecendo como resposta uma string correspondendo aos dados decifrados fornecidos durante a escrita no dispositivo, utilizando-se também da chave e do valor inicial fornecidos durante a carga do módulo.

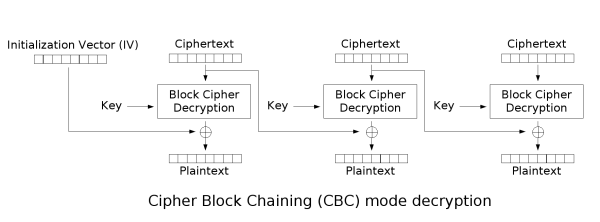
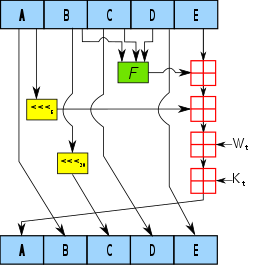


Figura 2: Diagrama do algoritmo de decifra AES em modo CBC.

Para os validar a cifragem e a decifragem realizadas, foi utilizado o site *http://aes.online-domain-tools.com/*, cujo algoritmo de padding é o mesmo utilizado no projeto, onde iserimos “0” a direita da palavra até que o bloco seja completo.

Para a operação de cálculo do hash foi utilizado o algoritmo SHA1, onde é retornada uma string correspondendo ao resumo criptográfico em hexadecimal dos dados fornecidos durante a escrita no dispositivo.



Funcionamento do SHA1:

Uma iteração dentro da função de compressão do algoritimo:

A, B, C ,D e E são palavras de 32 bits do estado;

F é uma função não-linear que varia;

<<< denota a rotação de bits à esquerda em n espaços, n varia para cada operação;

Wt é a palavra da mensagem expandida da rodada t;

Kt é a constante da rodada t;

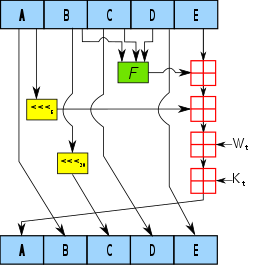
 denota uma adição módulo 232.

Figura 3: Diagrama do algoritmo de hash SHA-1.

Para o projeto, também foram implementadas no módulo de kernel MUTEX LOCKS, utilizados para bloquear o programa de testes em espaço de usuário caso o dispositivo /dev/crypto já esteja sendo utilizado por outro processo.

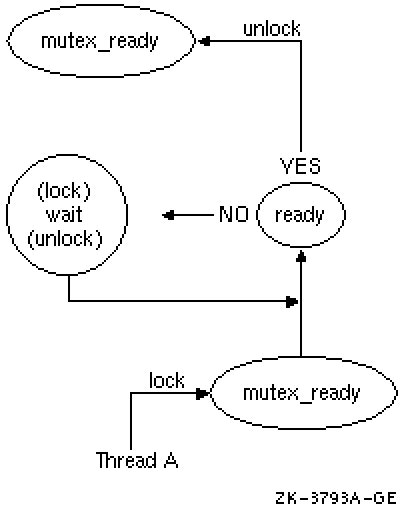


Figura 4: Diagrama do algoritmo do MUTEX LOCKS.

Por fim, o programa de testes abre o dispositivo do modulo, le a entrada do usuario e escreve os dados no dispositivo /dev/crypto os dados digitados pelo usuario. Em seguida, o programa le os dados retornados pelo modulo e os imprime no terminal.

Quanto a execucao do projeto, o carregamento do modulo é executado no seguinte formato:

insmod cryptochar.ko key=”010203040506070809A1A2A3A4A5A6A7” iv=”010203040506070809A1A2A3A4A5A6A7”

No programa teste, o primeiro caractere define a operação, sendo estas cifrar (c), decifrar (d) e calcular o hash (h) do dado informado.

Os dados de chave e valor de inicio são passados em formato de string, representados em hexadecimal (cada byte corresponde a dois dígitos hexa), enquanto o dado a sofrer a operacao é passado como plaintext.

**3.Resultados**

Primeiramente, foi realizado a compilacao do modulo e do programa de testes utilizando um arquivo makefile.

* Make do modulo e programa teste

Figura 5: Output do comando “make”

Em seguida, o modulo foi inserido com os valores de key = XXXXXXXXXXXXXXXX e iv = XXXXXXXXXXXXXXXXX.

* insmod do modulo passando os valores de key e iv

Figura 6: Output do Makefile passando os valores de iv e key.

Os dados foram verificados atraves do comando dmesg.

* Dmesg do modulo

Figura 7: Mensagens do kernel em sua insercao

Em seguida, foi executado o programa teste.

* executando programa teste

Figura 8: Chamada de execucao do programa teste

Primeiramente, foi testada a cifragem de dados.

* Cifragem input

Figura 9: Entrada de dados para cifragem

* cifragem output

Figura 10: Output da operação de cifra do módulo.

Em seguida, foram verificadas as mensagens de retorno do modulo para o kernel.

* dmesg do modulo após input

Figura 11: Mensagens do modulo para a criptografia.

O resultado obtido foi comparado com um algoritmo da internet.

* Teste de cifragem na internet

Figura 12: Comparacao de resultados de cifragem no site http://aes.online-domain-tools.com/.

Com isso, foi verificado o correto funcionamento da cifragem.

Em seguida, a decifragem foi testada.

* Decifragem input

Figura 13: Entrada de dados para decifragem.

* decifragem output

Figura 14: Output da operação de decifra do módulo.

Apos o pedido de decifragem, foram verificadas as mensagens do modulo para o kernel.

* Dmesg do modulo após output

Figura 15: Mensagens do modulo para a decifragem.

O resultado obtido foi comparado com um algoritmo da internet.

* teste de decifragem na internet

Figura 16: Comparacao de resultados de decifragem no site http://aes.online-domain-tools.com/

Com isso, foi verificado o correto funcionamento da cifragem.

Em seguida, o calculo do hash foi testado.

* Hash input

Figura 17: Entrada de dados para calculo do hash.

* hash output

Figura 18: Output da operação de hash do módulo.

O resultado obtido foi comparado com um algoritmo da internet.

* teste hash na internet

Figura 19: Comparacao de resultados de decifragem no site http://www.sha1-online.com/.

Com isso, foi verificado o correto funcionamento do hash.

Em seguida, foram testados os mutex locks.

* Teste Mutex (execucao de 2 programas teste ao mesmo tempo)

Figura 20: teste do MUTEX LOCKS no módulo.

Com isso, foi verificado o correto funcionamento do mutex.

**4.Conclusão**

O projeto funcionou como esperado, apresentando o correto funcionamento de suas funcoes.

Através deste projeto, foi possível familiarizar-se com os detalhes de implementação de um módulo de kernel e com o uso da API criptográfica do kernel Linux.

Através do estudo deste módulo, foram aprimorados os conceitos de implementação, compilação, instalação e testes de um novo módulo de kernel que realiza as funções de cifrar, decifrar e calcular o hash dos dados fornecidos pelo usuário. Também aprendeu-se sobre a utilizacao de outas funcoes do kernel, como scatterlists e mutexs.