Entrega: 10/10

Device file: /dev/crypto  
Módulo de Kernel: insmod cryptomodule.ko key=”secretkey”  
Operações:

Pode dar comandos pra ele cifrar, decifrar ou calcular o hash.  
Abre dispositivo, escreve o comando e quando for ler ele da o resultado da operação.  
Carrega módulo passa a key (chave criptografica).  
  
c string de dados -> cifra a string de dados usando AES e a chave key  
d string cifrada -> decifrar usando AES e a chave jet a “string cifrada”  
h string de dados -> calcula o resumo criptográfico (hash) usando SHA1  
  
Materiais complementares:  
1) tem exemplos (escreve comando, na leitura faz o resultado)  
2) ensina como criar um char device, escrita, leitura, que é o que teremos que fazer   
  
muito útil, muitos sistemas usam isso para cifrar e decifrar, pode-se fazer isso na aplicação, mas se fizer direto no kernel fica mais dificil de burlar(?).  
como escrever num device driver  
como ler  
entender o básico dos recursos do kernel

Descrição:  
  
Fazer:  
1) Módulo que cifre, decifre e calcula o HASH dos dados fornecidos.  
2) Implementar um programa em espaço de usuário para testar o módulo.  
  
1)  
O módulo deve ser capaz de receber requisições e enviar respostas através do arquivo de dispositivo **/dev/crypto**.  
  
Ao carregar o módulo de kernel, deve-se informar no parâmetro key a chave simétrica que será usada para cifrar e decifrar os dados, como no exemplo a seguir:   
**insmod cryptomodule.ko key=”secretkey”**Requisições devem ser: **operação dados**Operação: cifrar (c), decifrar (d) ou calcular hash (h).  
Dados: String com os dados para a operação. O caractere pode ser parte da string.  
  
Resposta: Ler no arquivo de dispositivo.  
 - Cifrar: String correspondendo aos dados cifrados com o algoritmo AES usando a chave fornecida durante a carga do módulo.  
 <https://www.kernel.org/doc/html/v4.12/crypto/api-samples.html#code-example-for-symmetric-key-cipher-operation>  
  
 - Decifrar: String correspondendo aos dados decifrados com algoritmo AES usando a chave.  
 - Resumo Criptográfico: String correspondendo ao resumo criptográfico hexadecimal dos dados fornecidos, usando SHA1.  
  
2)  
Para testar:  
  
Programa em espaço de usuário *(The benefit is that bugs do not lead to a kernel panic (a system crash) but to a simple crash (segmentation fault). Additionally, all tools for debugging like gdb can be used.)* que abra o arquivo de dispositivo, envie uma requisição fornecida pelo usuário (operação de escrita no arquivo de dispositivo) e exiba a resposta fornecida pelo dispositivo (operação de leitura).  
  
 dev\_open(): Called each time the device is opened from user space.

* dev\_read(): Called when data is sent from the device to user space.
* dev\_write(): Called when data is sent from user space to the device.
* dev\_release(): Called when the device is closed in user space.

Ver ebbchar.c.  
  
**O MÓDULO DE KERNEL E O PROGRAMA DEVEM SER COMPILADOS COM MAKEFILE.  
A API crypto do kernel nos dá diferentes chamadas de API para os seguintes tipos de cifras:**  
  
- Cifras simétricas  
- Cifras AEAD  
- RNG  
- Message digest  
- Interface em espaço de usuário  
  
**Procurando Cifras e Templates:**A API nos da implementacoes de blocos unicos cifrados e “message digests”. Em adição, a API crypto nos da vários templates que podem ser usado em conjunto com os blocos cifrados e “message digests”.   
  
Esses blocks cifrados ou as “message digests” podem ser usadas diretamente pelo “caller” ou invocadas junta com o template para fazer um “multi-block” criframento ou message digest com chave (keyed).  
  
Lista de criphers disponiveis:  
**cat** ***/proc/crypto***

### **Synchronous And Asynchronous Operation**

Quando usada operação síncronas da API, o caller invoca uma operação de cifragem que é executade de maneira sincronizada pelo kernel crypto API. Isso significa, que o caller espera até que a operação de cifra termine.  
  
Quando ela é assíncrona implica que a invocação de uma operação de cifra vai terminar praticamente instantaneamente. Ela ativa a operação de cifra mas não sinaliza quando está completa.

**Crypto API Cipher References And Priority**

Uma cifra é referenciada pelo caller como uma string com a seguinte semantica:  
  
template(single block cipher)  
template1(template2(single block cipher)))

### 

#### **Generic Block Cipher Structure**

Por exemplo, cbc(aes) é implementado com cbc.c, e aes-generic.c.

## **Developing Cipher Algorithms**

Cadastrando e Descadastrando Transformações:  
  
Há três tipos de funções de cadastramentos na API Crypto.  
Uma é usada para cadastrar uma transformação criptográfica genérica, enquanto as outras duas são específicas para transformação de HASH e Compressão.  
  
Antes de discutir cadastramento, a estrutura de dados a ser preenchida precisa ser considerada: struct crypto\_alg.  
  
Essa funcao de cadastramento genérica pode ser encontrada em /include/linux/crypto.h.  
Cadastrar e descadastrar:

**int** **crypto\_register\_alg**(**struct** crypto\_alg **\***alg);  
**int** **crypto\_register\_algs**(**struct** crypto\_alg **\***algs, **int** count);

**int** **crypto\_unregister\_alg**(**struct** crypto\_alg **\***alg);  
**int** **crypto\_unregister\_algs**(**struct** crypto\_alg **\***algs, **int** count);

Se retornar 0 então é sucesso. < 0 deu erro.

**crypto\_alg** é um array do tamanho de **count**. Essas função simplesmente fazem loop no array e cadastram ou nao cada algoritmo individualmente. O caller nao tem como saber qual implementacao de cipher foi cadastrada com sucesso.

### **Single-Block Symmetric Ciphers [CIPHER]**

Exemplo: AES.  
  
É a mais simples de todas as transformações, que seria o tipo de cifra utilizada em cifrar simétricas. Vai operar em apenas um bloco por vez e não há dependência entre os blocos.

**Pulo para:   
  
User Space Interface**

O espaço de usuário chama API’s de maneira totalmente síncrona.  
  
The kernel crypto API is accessible from user space. Currently, the following ciphers are accessible:

* Message digest including keyed message digest (HMAC, CMAC)
* Symmetric ciphers ← ESSA É A AES
* AEAD ciphers
* Random Number Generators

A interface é por via de SOCKET usando o tipo AF\_ALG.   
Em adição, a opção setsockopt é SOL\_ALG.  
  
**#ifndef AF\_ALG**  
**#define AF\_ALG 38**  
**#endif**  
**#ifndef SOL\_ALG**  
**#define SOL\_ALG 279**  
**#endif**

Uma cifra é acessado com o mesmo nome da API do kernel.  
Para interagir com a API crypto do kernel, um socket deve ser criado pela aplicação do usuário. O usuário invoca a operacao com send()/write(). O resultado é obtido com read()/recv().  
  
*To initialize the socket interface, the following sequence has to be performed by the consumer:*

1. *Create a socket of type AF\_ALG with the struct sockaddr\_alg parameter specified below for the different cipher types.*
2. *Invoke bind with the socket descriptor*
3. *Invoke accept with the socket descriptor. The accept system call returns a new file descriptor that is to be used to interact with the particular cipher instance. When invoking send/write or recv/read system calls to send data to the kernel or obtain data from the kernel, the file descriptor returned by accept must be used.*

Ler: [**Symmetric Cipher API**](https://kernel.readthedocs.io/en/sphinx-samples/crypto-API.html#symmetric-cipher-api)