struct tcrypt\_result {  
 struct completion completion;  
 int err;  
};  
  
/\* tie all data structures together \*/  
struct skcipher\_def {  
 struct scatterlist sg;  
 struct crypto\_skcipher \*tfm;  
 struct skcipher\_request \*req;  
 struct tcrypt\_result result;  
};  
  
/\* Callback function \*/  
static void test\_skcipher\_cb(struct crypto\_async\_request \*req, int error)  
{  
 struct tcrypt\_result \*result = req->data;  
  
 if (error == -EINPROGRESS)  
 return;  
 result->err = error;  
 complete(&result->completion);  
 pr\_info("Encryption finished successfully\n");  
}  
  
/\* Perform cipher operation \*/  
static unsigned int test\_skcipher\_encdec(struct skcipher\_def \*sk,  
 int enc)  
{  
 int rc = 0;  
  
 if (enc)  
 rc = crypto\_skcipher\_encrypt(sk->req);  
 else  
 rc = crypto\_skcipher\_decrypt(sk->req);  
  
 switch (rc) {  
 case 0:  
 break;  
 case -EINPROGRESS:  
 case -EBUSY:  
 rc = wait\_for\_completion\_interruptible(  
 &sk->result.completion);  
 if (!rc && !sk->result.err) {  
 reinit\_completion(&sk->result.completion);  
 break;  
 }  
 default:  
 pr\_info("skcipher encrypt returned with %d result %d\n",  
 rc, sk->result.err);  
 break;  
 }  
 init\_completion(&sk->result.completion);  
  
 return rc;  
}  
  
/\* Initialize and trigger cipher operation \*/  
static int test\_skcipher(void)  
{  
 struct skcipher\_def sk;  
 struct crypto\_skcipher \*skcipher = NULL;  
 struct skcipher\_request \*req = NULL;  
 char \*scratchpad = NULL;  
 char \*ivdata = NULL;  
 unsigned char key[16]; **//####### SERÁ CRIADO PELO USUARIO -> SE TIVER MAIS QUE O TAMANHO DEFINIDO, CORTA O VETOR. SENÃO, ADICIONA “0” ATÉ DAR O TAMANHO**  
 int ret = -EFAULT;  
  
 skcipher = crypto\_alloc\_skcipher("cbc-aes-aesni", 0, 0); **//####### ERRADO -> VER LINHA ABAIXO**  
 skcipher = crypto\_alloc\_skcipher("aes", 0, 0); **//####### PRECISA TESTAR SE ISSO DÁ CERTO**

if (IS\_ERR(skcipher)) {  
 pr\_info("could not allocate skcipher handle\n");  
 return PTR\_ERR(skcipher);  
 }  
  
 req = skcipher\_request\_alloc(skcipher, GFP\_KERNEL);  
 if (!req) {  
 pr\_info("could not allocate skcipher request\n");  
 ret = -ENOMEM;  
 goto out;  
 }  
  
 skcipher\_request\_set\_callback(req, CRYPTO\_TFM\_REQ\_MAY\_BACKLOG,  
 test\_skcipher\_cb,  
 &sk.result);  
  
 /\* AES 128 with random key \*/  
 get\_random\_bytes(&key, 16); **//####### SERÁ CRIADO PELO USUARIO**  
 if (crypto\_skcipher\_setkey(skcipher, key, 16)) {  
 pr\_info("key could not be set\n");  
 ret = -EAGAIN;  
 goto out;  
 }  
  
 /\* IV will be random \*/  
 ivdata = kmalloc(16, GFP\_KERNEL);  
 if (!ivdata) {  
 pr\_info("could not allocate ivdata\n");  
 goto out;  
 }  
 get\_random\_bytes(ivdata, 16); **//####### É UM VALOR FIXO, NÃO SERÃO DADOS ALEATÓRIOS**  
  
 /\* Input data will be random \*/  
 scratchpad = kmalloc(16, GFP\_KERNEL);  
 if (!scratchpad) {  
 pr\_info("could not allocate scratchpad\n");  
 goto out;  
 }  
 get\_random\_bytes(scratchpad, 16); **//####### SERÁ CRIADO PELO USUARIO -> DESCONSIDERAR**  
  
 sk.tfm = skcipher;  
 sk.req = req;  
  
 /\* We encrypt one block \*/  
 sg\_init\_one(&sk.sg, scratchpad, 16);  
 skcipher\_request\_set\_crypt(req, &sk.sg, &sk.sg, 16, ivdata);  
 init\_completion(&sk.result.completion);  
  
 /\* encrypt data \*/  
 ret = test\_skcipher\_encdec(&sk, 1); **//####### SEGUNDO PARÂMETRO = 1 -> CIFRAR; 2 -> DECIFRAR**  
 if (ret)  
 goto out;  
  
 pr\_info("Encryption triggered successfully\n");  
  
out:  
 if (skcipher)  
 crypto\_free\_skcipher(skcipher);  
 if (req)  
 skcipher\_request\_free(req);  
 if (ivdata)  
 kfree(ivdata);  
 if (scratchpad)  
 kfree(scratchpad);  
 return ret;  
}

<https://www.kernel.org/doc/html/v4.12/crypto/api-samples.html#code-example-for-symmetric-key-cipher-operation>