**QEMU**

virtio\_error() // 调用此函数做debug，会导致设备broken；virtio.c

GCC\_FMT\_ATTR{ //2529行左右

vdev->broken = true;

}

日志输出改用：error\_report 、info\_report

config-host.mak

CONFIG\_VHOST\_CRYPTO=y

config-host.h

#define CONFIG\_VHOST\_CRYPTO 1

添加日志

#include "qemu/error-report.h"

info\_report("zhaorui--%s, call", \_\_FUNCTION\_\_);

增加随机数算法

CryptoDevBackendClass{ // cryptodev.h

do\_rng\_op

}

virtio\_crypto\_device\_realize{// virtio-crypto.c

vcrypto->vqs[i].dataq\_bh = qemu\_bh\_new(virtio\_crypto\_dataq\_bh, &vcrypto->vqs[i]);

}

---------------------------------

修改内核头文件virtio\_crypto.h

virtio\_crypto\_op\_ctrl\_req //控制队列命令

virtio\_crypto\_op\_data\_req{ //数据队列请求

virtio\_crypto\_op\_header{//加解密相关操作

opcode,//添加VIRTIO\_CRYPTO\_GET\_RNG

}

------------------------------

virtio\_crypto\_dataq\_bh //中断下半部

virtio\_crypto\_handle\_dataq

virtio\_crypto\_handle\_request{ // virtio-crypto.c，增加随机数opcode

VirtQueueElement \*elem = &request->elem

iov\_to\_buf(virtio\_crypto\_op\_data\_req req <= elem->out\_sg) //vring拷贝到结构体

cryptodev\_backend\_crypto\_operation // cryptodev.c

}

------------------

qcrypto\_cipher\_new //创建内置算法引擎，cipher.c，builtin模式使用

qcrypto\_cipher\_lib\_driver // cipher-gcrypt.c，库模式，如openssl等

qcrypto\_cipher\_afalg\_driver // cipher-afalg.c，内核接口模式

================================

**虚拟机内部驱动**

**内核算法API**

/linux/include/crypto/hash.h

rng.h

virtio\_crypto前端驱动

/linux/drivers/crypto/virtio/ virtio\_crypto.ko

Linux系统的驱动默认存放目录为： /lib/modules/$(uname -r)/kernel/drivers/

virtio\_crypto. ko

/lib/modules/4.15.0-29-generic/kernel/drivers/crypto/virtio

struct ablkcipher\_alg ablkcipher; //异步block cipher算法

struct blkcipher\_alg blkcipher; //同步block cipher算法

struct cipher\_alg cipher;

struct compress\_alg compress;

**加载前端驱动**

先加载依赖

sudo insmod ~/work/tmp/linux-4.15/crypto/crypto\_engine.ko

或许还需要加载：

sudo insmod ~/work/linux-4.15/drivers/virtio/virtio\_ring.ko

sudo insmod ~/work/tmp/linux-4.15/drivers/virtio/virtio\_pci.ko

再加载virtio\_crypto

sudo insmod ~/work/linux-4.15/drivers/crypto/virtio/virtio\_crypto.ko

内核加密框架中，使用结构crypto\_alg来描述一个算法，每一个算法(实例)相当于一个类，在实际的使用环境中，在每一次运行调用时，需要为它分配一个对像，在内核加密框架中，这个“对像”被称为transform（简称tfm）。transform意味“变换”，可能译为“蜕变”更为合适。tfm是加密框架中一个极为重要的概念，它由结构crypto\_tfm描述：

struct crypto\_tfm {

        u32 crt\_flags;

        union {

                struct ablkcipher\_tfm ablkcipher;

                struct aead\_tfm aead;

                struct blkcipher\_tfm blkcipher;

                struct cipher\_tfm cipher;

                struct hash\_tfm hash;

                struct ahash\_tfm ahash;

                struct compress\_tfm compress;

                struct rng\_tfm rng;

        } crt\_u;

        void (\*exit)(struct crypto\_tfm \*tfm);

        struct crypto\_alg \*\_\_crt\_alg;

        void \*\_\_crt\_ctx[] CRYPTO\_MINALIGN\_ATTR;

};

结构定义了一个名为crt\_u的联合体，以对应每种算法的tfm的具体操作，例如加密/解密，求hash，压缩/解压等，加密框架引入了一组名为xxx\_tfm的结构封装。

对于算法的实始化，其核心功能就是分配一个tfm，并设置其上下文环境，例如密钥等参数，然后初始化上述struct xxx\_tfm结构。

如crypto\_alloc\_hash，crypto\_alloc\_rng等等

crypto\_init\_ops负责初始化tfm的选项，对于一个真正的算法（例如md5、dst）和一个伪算法（指由模版动态分配的，如hmac(xxx), authenc(xxx,xxx)），它们的初始化过程是截然不同的。一个伪算法，它都设置了其所属的类型cra\_type，例如，对于hmac(xxx)而言，它指向了crypto\_hash\_type。这样，初始化时，实质上调用的是其所属类型的init函数

==========================

内核cryptodev模块(/dev/crypto)，调用内核算法流程：

1）初始化tfm

crypto\_create\_session{ //cryptodev /ioctl.c

cryptodev\_cipher\_init -> cryptodev\_crypto\_alloc\_blkcipher-> crypto\_alloc\_ablkcipher

}

内核加密框架中，所有的hash算法用结构shash\_alg描述，它封装了crypto\_alg和hash函数所需的一些特别的元素。

====================**Guest前端virtio\_crypto driver** ====================

系统头文件：

vim /lib/modules/$(shell uname -r)/build/ include/uapi/linux/ virtio\_crypto.h

**初始化：**

virtcrypto\_init\_vqs

virtcrypto\_find\_vqs //data\_vq设置virtcrypto\_dataq\_callback，异步接收返回数据

virtio\_crypto\_ablkcipher\_crypt\_req//注册到crypto\_engine->cipher\_one\_request

\_\_virtio\_crypto\_ablkcipher\_do\_req //cpu\_to\_le32，写虚拟硬件IO

virtio\_crypto\_algs\_register //注册算法

struct crypto\_alg virtio\_crypto\_algs{

.setkey = virtio\_crypto\_ablkcipher\_setkey,

.decrypt = virtio\_crypto\_ablkcipher\_decrypt,

.encrypt = virtio\_crypto\_ablkcipher\_encrypt,

}

调用：

控制队列ctrl queue使用同步调用，数据队列dataqueue使用异步调用

1)setkey/ decrypt/ encrypt

2)virtio\_crypto\_ablkcipher\_setkey/ virtio\_crypto\_ablkcipher\_decrypt/ virtio\_crypto\_ablkcipher\_..

3)virtio\_crypto\_ablkcipher\_setkey{ //同步调用

virtio\_crypto\_alg\_ablkcipher\_init\_session()// 写虚拟IO地址, ring

}

virtio\_crypto\_ablkcipher\_decrypt

virtio\_crypto\_ablkcipher\_encrypt{ //异步调用

virtio\_crypto\_dataq\_sym\_callback//注册请求完成函数

crypto\_transfer\_cipher\_request\_to\_engine //发送请求到engine即返回

}

4) engine收到请求，执行回调，填充desc ring，通知host：

virtio\_crypto\_ablkcipher\_crypt\_req{

\_\_virtio\_crypto\_ablkcipher\_do\_req{ //写虚拟IO地址, ring

virtqueue\_kick

}

virtqueue\_kick

}

5）virtqueue收到host返回数据通知后，回调：

virtcrypto\_dataq\_callback{

virtqueue\_get\_buf()//此时返回数据已经存入in\_sg指向内存，释放desc ring

virtio\_crypto\_dataq\_sym\_callback()

}

virtcrypto\_find\_vqs{

callbacks[i] = virtcrypto\_dataq\_callback;//注册异步调用，完成回调函数

}

===============Guest 用户层接口Cryptodev-linux=================

Cryptodev-linux

使用/dev/crypto做通信，此模块未实现取随机数等操作，只实现了对称算法和HASH、HMAC算法，不如原生netlink接口的libkcapi库实现的全面，性能比netlink接口稍高

下载地址：

https://github.com/cryptodev-linux/cryptodev-linux

cryptodev-linux-1.10.tar.gz

**安装依赖：**

sudo apt-get install libelf-dev

sudo apt-get install openssl

sudo apt-get install libssl-dev // libssl1.1\_1.1.0g

(centos是yum install openssl-devel)

编译、安装

make

sudo insmod cryptodev.ko

sudo insmod ~/work/cryptodev-linux-cryptodev-linux-1.10/cryptodev.ko

//sudo make install // insmod，并且将头文件安装到系统目录

-----------查看dmesg输出错误信息--------------->

sudo insmod cryptodev.ko时的错误

cryptodev: loading out-of-tree module taints kernel.

cryptodev: module verification failed: signature and/or required key missing - tainting kernel

cryptodev: driver 1.10 loaded.

解决：

似乎只是警告，lsmod可以看到已经加载模块，并不影响使用

网上答案：

It seems like the vendor of your system has enabled kernel module signature verification on your kernel which means it won't load any module that the vendor hasn't signed. In other words, your patched module isn't signed (properly) and the kernel will refuse to load it.

I suggest you contact your vendor. There may be an option somewhere on your platform to disable signature checking. Otherwise, your vendor may be able to sign the module for you. You might even have the key and the details of the signature verification algorithm and can sign it yourself.

<-----------查看dmesg输出错误信息---------------

lsmod //查看模块加载

ls /dev/crypto //查看模块生成目录

将模块加入到随系统启动：

-------------->

In most systems this can be done as:

# echo "cryptodev" >>/etc/modules

or in systemd-enabled systems:

# echo "cryptodev" > /etc/modules-load.d/cryptodev.conf

<----------------

Linux系统的驱动默认存放目录为： /lib/modules/$(uname -r)/kernel/drivers，因此如果希望系统可以启动时自动加载驱动必须首先把驱动放到该目录下面，执行完此步操作以后，再执行 depmod 命令更新驱动依赖关系文件 modules.dep，这一步一定要执行。之后我们重启系统，就可以发现驱动会自动加载了。

cp cryptodev.ko /lib/modules/$(uname -r)/kernel/drivers/crypto/

depmod

----------------------------------------------

测试

make check //会编译cryptodev-linux-1.10\tests目录下测试程序，然后执行：

./cipher

./hmac

./async\_cipher

./async\_hmac

./cipher-aead-srtp

./cipher-gcm

./cipher-aead

或者进入tests目录，执行：

make

编译出测试程序，但不运行

make check没有日志输出，可以在tests目录单独执行某个测试程序，加上debug参数，如：

./cipher debug //默认调用内核集成的软算法，输出如下：

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver cbc(aes-generic)

AES Test passed

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver cbc(aes-generic)

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver cbc(aes-generic)

Test passed

当加载virtio\_crypto模块时，则变成调用virtio\_crypto提供算法

./cipher debug //输出如下：

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver virtio\_crypto\_aes\_cbc

AES Test passed

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver virtio\_crypto\_aes\_cbc

requested cipher CRYPTO\_AES\_CBC, got cbc(aes) with driver virtio\_crypto\_aes\_cbc

Test passed

注意：

不可以直接在tests执行make，需要在根目录执行make check，否则没有输出

----------------------------------------------

使用sysctl tool设置cryptodev模块 的debug选项：

获取当前值

$sysctl ioctl.cryptodev\_verbosity

$ioctl.cryptodev\_verbosity = 0

设置新值

$sysctl ioctl.cryptodev\_verbosity=3

$ioctl.cryptodev\_verbosity = 3

使用方法

a) int cfd = open("/dev/crypto");

b) Fill in common struct cryptodev ctx

c) Fill in struct crypt op

d) Pass struct crypt op into kernel via ioctl()

e) Retrieve results

f) close(cfd);

在openssl中使用cryptodev做引擎，编译时：

./config -fPIC --prefix=/usr/local/openssl enable-shared

./config -t

生成makefile

在makefile CFLAGS增加配置：

-DHAVE\_CRYPTODEV 和 -DUSE\_CRYPTODEV\_DIGESTS flags

./openssl version -a //查看编译选项中是否有HAVE\_CRYPTODEV&USE\_CRYPTODEV\_DIGESTS

注：此方法不准

使用openssl调用engine测试

./openssl speed -evp aes-128-cbc -engine cryptodev

如果openssl编译未使用-DHAVE\_CRYPTODEV 和 -DUSE\_CRYPTODEV\_DIGESTS，则会报错，输出错误信息如下：

invalid engine "cryptodev"

139945467314624:error:25066067:DSO support routines:dlfcn\_load:could not load the shared library:../crypto/dso/dso\_dlfcn.c:113:filename(/usr/lib/x86\_64-linux-gnu/engines-1.1/cryptodev.so): /usr/lib/x86\_64-linux-gnu/engines-1.1/cryptodev.so: cannot open shared object file: No such file or directory

139945467314624:error:25070067:DSO support routines:DSO\_load:could not load the shared library:../crypto/dso/dso\_lib.c:161:

139945467314624:error:260B6084:engine routines:dynamic\_load:dso not found:../crypto/engine/eng\_dyn.c:414:

139945467314624:error:2606A074:engine routines:ENGINE\_by\_id:no such engine:../crypto/engine/eng\_list.c:339:id=cryptodev

139945467314624:error:25066067:DSO support routines:dlfcn\_load:could not load the shared library:../crypto/dso/dso\_dlfcn.c:113:filename(libcryptodev.so): libcryptodev.so: cannot open shared object file: No such file or directory

139945467314624:error:25070067:DSO support routines:DSO\_load:could not load the shared library:../crypto/dso/dso\_lib.c:161:

139945467314624:error:260B6084:engine routines:dynamic\_load:dso not found:../crypto/engine/eng\_dyn.c:414:

------------------------------------>

正常输出：

engine "cryptodev" set.

Doing aes-128-cbc for 3s on 16 size blocks: 168241 aes-128-cbc's in 0.04s

Doing aes-128-cbc for 3s on 64 size blocks: 163675 aes-128-cbc's in 0.05s

Doing aes-128-cbc for 3s on 256 size blocks: 148530 aes-128-cbc's in 0.04s

Doing aes-128-cbc for 3s on 1024 size blocks: 108843 aes-128-cbc's in 0.04s

Doing aes-128-cbc for 3s on 8192 size blocks: 30926 aes-128-cbc's in 0.01s

Doing aes-128-cbc for 3s on 16384 size blocks: 16668 aes-128-cbc's in 0.01s

OpenSSL 1.1.0g 2 Nov 2017

built on: reproducible build, date unspecified

options:bn(64,64) rc4(16x,int) des(int) aes(partial) idea(int) blowfish(ptr)

compiler: gcc -DDSO\_DLFCN -DHAVE\_DLFCN\_H -DNDEBUG -DOPENSSL\_THREADS -DOPENSSL\_NO\_STATIC\_ENGINE -DOPENSSL\_PIC -DOPENSSL\_IA32\_SSE2 -DOPENSSL\_BN\_ASM\_MONT -DOPENSSL\_BN\_ASM\_MONT5 -DOPENSSL\_BN\_ASM\_GF2m -DSHA1\_ASM -DSHA256\_ASM -DSHA512\_ASM -DRC4\_ASM -DMD5\_ASM -DAES\_ASM -DVPAES\_ASM -DBSAES\_ASM -DGHASH\_ASM -DECP\_NISTZ256\_ASM -DPADLOCK\_ASM -DPOLY1305\_ASM -DOPENSSLDIR="\"/usr/local/openssl/ssl\"" -DENGINESDIR="\"/usr/local/openssl/lib/engines-1.1\"" -fPIC -Wa,--noexecstack

The 'numbers' are in 1000s of bytes per second processed.

type 16 bytes 64 bytes 256 bytes 1024 bytes 8192 bytes 16384 bytes

aes-128-cbc 67296.40k 209504.00k 950592.00k 2786380.80k 25334579.20k 27308851.20k

<-----------------------------------------------

查看内核注册算法

cat /proc/crypto

name : cbc(aes)

driver : virtio\_crypto\_aes\_cbc

module : virtio\_crypto

priority : 150

refcnt : 1

selftest : passed

internal : no

type : ablkcipher

async : yes

blocksize : 16

min keysize : 16

max keysize : 32

ivsize : 16

geniv : <default>

===============Guest 用户层接口libkcapi=================

The Linux kernel exports a Netlink interface of type AF\_ALG to allow user space to utilize the kernel crypto API.

libkcapi uses this Netlink interface and exports easy to use APIs so that a developer does not need to consider the low-level Netlink interface handling.

The library does not implement any cipher algorithms. All consumer requests are sent to the kernel for processing. Results from the kernel crypto API are returned to the consumer via the library API.

The kernel interface and therefore this library can be used by unprivileged processes.

The focus during the development of this library is put on speed. This library does not perform any memcpy for processing the cryptographic data! The library uses scatter / gather lists to eliminate the need for moving data around in memory.

主页：

http://www.chronox.de/libkcapi.html

下载地址：

https://github.com/smuellerDD/libkcapi

http://www.chronox.de/libkcapi/libkcapi-1.1.4.tar.xz

编译、安装：

tar -xJf libkcapi-1.1.4.tar.xz

sudo apt-get install libtool autoconf

autoreconf -i

./configure --prefix=/usr/local --enable-kcapi-test --enable-kcapi-rngapp --enable-kcapi-speed --enable-kcapi-hasher --enable-kcapi-encapp --enable-kcapi-dgstapp

make

测试：

cd libkcapi-1.1.4/bin

//使用jitterentropy\_rng引擎，取100字节随机数，格式化为HEX

./kcapi-rng -b 100 --hex -n jitterentropy\_rng

./kcapi-rng -b 100 --hex -n stdrng

./kcapi-rng -b 50 --hex -n zj\_rng

=========================================================

调试内核virtio\_crypto模块：

卸载内核相关模块：

sudo rmmod virtio\_pci //启动VM时不要加载网络参数，否则无法卸载

sudo rmmod virtio\_ring

安装依赖模块：

sudo insmod ~/work/tmp/linux-4.15/crypto/crypto\_engine.ko //手动编译的内核,未安装

sudo insmod ~/work/linux-4.15/drivers/virtio/virtio\_ring.ko //修改后的模块

sudo insmod ~/work/tmp/linux-4.15/drivers/virtio/virtio\_pci.ko

安装修改后的相关模块：

sudo insmod ~/work/linux-4.15/drivers/crypto/virtio/virtio\_crypto.ko

查看是否注册成功

cat /proc/crypto

手动安装模块的方式，经常不能注册/proc/crypto成功，原因：

vhost模式，host需要先加载vhost.ko和vhost\_secard.ko，如果忘记，guest则不能注册成功

使用替换系统自带ko方式，发现未生效：

cp ~/work/linux-4.15/drivers/virtio/virtio\_ring.ko /lib/modules/4.15.0/kernel/drivers/virtio/

sudo cp virtio\_ring.ko /lib/modules/4.15.0/kernel/drivers/virtio/

就算将/lib/modules/4.15.0/kernel/drivers/virtio/ virtio\_ring.ko改名，系统重启动后，lsmod也还是正常加载了virtio\_ring模块（不知道从哪里加载的，find /未发现其他地方有virtio\_ring.ko，可能是已经编译到kernel内部？？？？）

======================Guest通知host流程=======================

virtio\_crypto\_rng\_random

virtqueue\_kick

virtqueue\_kick\_prepare

vring\_need\_event = needs\_kick = 1

virtqueue\_notify

Guest获取Host返回数据流程：

virtio\_crypto\_rng\_random

virtqueue\_get\_buf -> virtqueue\_get\_buf\_ctx

more\_used => vq->last\_used\_idx != vq->vring.used->idx

Guest 收到irq中断时的调用路径，Call Trace:

[ 281.929528] <IRQ>

[ 281.929532] virtcrypto\_dataq\_callback+0xff/0x120 [virtio\_crypto]

[ 281.929534] vring\_interrupt+0x37/0x80 [virtio\_ring]

[ 281.929537] \_\_handle\_irq\_event\_percpu+0x44/0x1a0

[ 281.929538] handle\_irq\_event\_percpu+0x32/0x80

[ 281.929539] handle\_irq\_event+0x3b/0x60

[ 281.929540] handle\_edge\_irq+0x7c/0x190

[ 281.929542] handle\_irq+0x20/0x30

[ 281.929543] do\_IRQ+0x46/0xd0

[ 281.929545] common\_interrupt+0x9f/0x9f

[ 281.929546] </IRQ>

================================================================