Gemini 3755 标准密码卡
用户手册

文档版本：v1.0

Serica® 赛芯半导体
SERICA TECHNOLOGY

目 录

1. 概述

1.1. 本文目的

本文档介绍了 Gemini 3755 标准密码卡的规格和特性，也提供了关于此密码卡的使用配置指南以及安装指南。

2. 产品介绍

Gemini 3755 标准密码卡产品面向高负载数据中心网络安全、通信安全及存储安全应用加密运算需求提供一站式解决方案。

2.1. 特色概述

- 支持对称算法 ARC4、DES/3DES（ECB/CBC/CTR）、AES（ECB/CBC/CTR/GCM）/SM4（ECB/CBC/GCM）
- 支持杂凑算法 SHA1/256/384/512/SM3
- 支持公钥算法 RSA1024/RSA2048/SM2
- 支持 SR-IOV，最多支持 32 个 VF
- 主要面向服务器、加密机等对算法性能要求较高的应用场景中，具有高性能、可编程等特性。

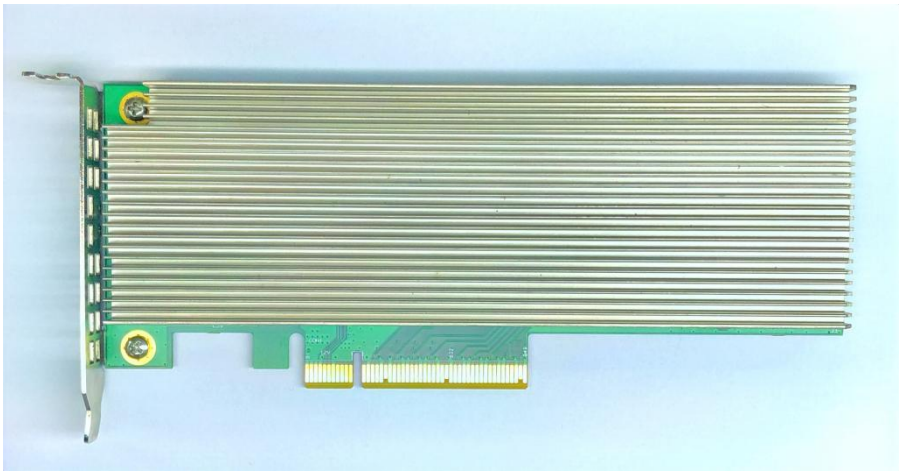
2.2. 规格参数

参数	描述
总体	
软件支持	· GM0018 规范 SDF 接口: Linux* and OpenSSL
产品功耗	· ~30W
硬件设计	· 产品各项电气指标兼容 PCI Express Card Electromechanical Specification, Revision 3.0 标准 · 主动式散热方案设计
工作温度	· 0°C to 45°C

产品尺寸(H x L 板卡)	· 68.9mm x 166.65mm
接口	· 适配 PCIe Gen2/3 x 8 插槽
核心处理器(MCU)	
功耗	· <25W
封装尺寸	· 167.25*69 mm
	·
性能指标	
国密算法支持	· SM1 性能 18Gbps；
	· SM2 签名 228000 次/秒、验签 108000 次/秒；
	· SM3 性能 19Gbps；
	· SM4 性能 25Gbps；
国际算法支持	· RSA1024 签名 92000 次/秒、验签 427000 次/秒；
	· RSA2048 签名 12600 次/秒、验签 410000 次/秒；
	· SHA 性能 20Gbps
	· AES 性能 24.5Gbps
随机数产生	· 3.4Gbps

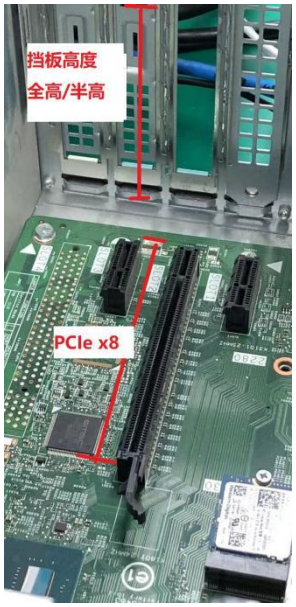
3. 硬件安装指南

3.1. 硬件形态



3.2. 硬件安装

3.2.1. 确认硬件主机插槽



3.2.2. 安装加密卡

4. 驱动安装及测试流程

4.1. 确认加密卡被识别

lspci -d 1dab:7002 -vvv（虚拟机 pci 设备号 1dab:8001）

如卡正确安装，显示如下

```
[root@localhost driver_3755_02]# lspci -d 1dab:7002
01:00.0 Encryption controller: Device 1dab:7002 (rev 01)
```

4.2. 编译软件

```
# tar xf driver_3755_02_v2.0.3.8.tar.gz
# cd driver_3755_02_v2.0.3.8
# ./build.sh
```

正常结束显示如下

```
test -d '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/logs' \
       || mkdir -p '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/logs'
test -d '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules' \
       || mkdir -p '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules'
test ! -f '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules/ngx_ssl_engine_rsp_module.so' \
       || mv '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules/ngx_ssl_engine_rsp_module.so' \
             '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules/ngx_ssl_engine_rsp_module.so.old'
cp objs/ngx_ssl_engine_rsp_module.so '/root/driver_3755_02_ssl_engine_1.0.0.0/out/nginx/modules/ngx_ssl_engine_rsp
make[1]: Leaving directory '/root/driver_3755_02_ssl_engine_1.0.0.0/nginx/asynch_mode_nginx'
nginx version: nginx/1.16.1

 Nginx build success.
```

4.3. 驱动加载并确认

lsmod | grep rsp

正常加载如下显示

```
[root@localhost driver_3755_02]# lsmod | grep rsp
rsp_cmdq_drv           22431  0
rsp_drv_pf             13562  1
rsp_ctrl_drv          245699  2 rsp_drv_pf,rsp_cmdq_drv
```

如果没有加载或机器重启后，需运行如下命令重新加载

./load_driver.sh

4.4. 基本算法测试

./test.sh

等待一段时间后，出现如下屏幕，开始执行测试

```
[root@localhost driver_3755_02]# ./test.sh

2023-12-01 11:12:05: DEV-1:01:00.0 / Total Devices: 1;

Async Mode [ 512 1024 2048 4096 ] Performance Test Start.


Config: pub p-t:[8*2], rpu p-t:[8*2]

SM2-KG                          : 305635.093750(tps),
SM2-KP                          : 86362.515625(tps),
SM2-ENC                         : 55263.093750(tps),
SM2-DEC                         : 67298.617188(tps),
SM2-SIGN                        : 294721.187500(tps),
SM2-VERIFY                      : 108271.539062(tps),
RSA-1024-ENC                    : 618373.937500(tps),
RSA-1024-DEC                    : 92230.328125(tps),
RSA-1024-DEC_CRT                : 261279.421875(tps),
RSA-1024-SIGN                   : 92212.203125(tps),
RSA-1024-SIGN_CRT               : 261302.781250(tps),
RSA-1024-VERIFY                 : 617538.000000(tps),
```

4.5. OpenSSL 和 Nginx 测试

```
# 解压并编译 openssl 扩展包，
 tar czf driver_3755_02_ssl_engine_1.0.0.0.tar.gz
driver_3755_02_ssl_engine_1.0.0.0
 cd  driver_3755_02_ssl_engine_1.0.0.0

# 修改配置文件里面的路径，指向上面的驱动目录
 vi rsp_ssl_engine.conf

... ...
export RSP_BASIC_PKG_PATH="/root/driver_3755_02_v2.0.3.8"
... ...

# 编译 openssl 扩展包
./build.sh
```

4.5.1. OpenSSL

```
# 执行性能测试脚本
./test/test_ssl_speed.sh
# 参数 1：SSL 引擎模式，支持 openssl 和 gmssl
# 参数 2：核心数（进程数），0 表示同时测试 1，2，4，8 核心数
# 参数 3：算法类型，all 表示测试全部已支持算法
# 参数 4：SSL 算法类型，hw 表示使用硬件引擎，sw 表示使用 SSL 软算法
```

参数 3 支持的算法种类:

```
Supported algo list:
  [sm3, sha1, sha256, sha384, sha512]
  [aes-128-cbc, aes-128-gcm, aes-256-cbc, aes-256-gcm, sm4-cbc, sm4-ctr]
  [rsa2048, sm2-encrypt, sm2-sign, ecdsa-p256, ecdh-p256]
```

例子:

```
./test/test_ssl_speed.sh openssl 8 sm4-cbc hw
```

```
SSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -
-DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM
evp             785477.59k  1517921.94k  2812614.72k  3233355.04k  3304536.01k  3360501.67k

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

evp 分别是包大小为 512， 1k， 2k， 4k，8k，16k 的性能，单位 KB/s

./test/test_ssl_speed.sh openssl 8 sm2_sign hw

```
-DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM
                              sign    verify    sign/s verify/s
 256 bits ecdsa (sm2)   0.0000s   0.0000s 270906.7 121668.6

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4.5.2. Nginx

nginx 性能测试脚本分为服务端和客户端，需要在两个控制台界面分别运行

1）服务端：
./test/nginx_ab_server.sh
参数 1：算法套件，支持 [rsa-rsa/ecdhe-ecdsa/ecdhe-rsa/ecc-sm2]
参数 2：核心数（进程数），0 表示同时测试 1，2，4，8 核心数
参数 3：SSL 算法类型，hw 表示使用硬件引擎，sw 表示使用 SSL 软算法
参数 4：SSL 引擎模式，支持 openssl 和 gmssl

下图时服务端启动输出

```
[root@localhost driver_3755_02_ssl_engine_1.0.0.0]# ./test/nginx_ab_server.sh rsa-rsa 8 hw openssl
Server Software:        nginx/1.16.1
Server Hostname:        localhost
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,AES256-GCM-SHA384,2048,256
Document Path:          /index.html
HTML transferred:       144 bytes
2024-06-19 15:27:36 nginx performance: 0
2024-06-19 15:27:37 nginx performance: 0
2024-06-19 15:27:38 nginx performance: 0
2024-06-19 15:27:39 nginx performance: 0
2024-06-19 15:27:40 nginx performance: 0
2024-06-19 15:27:41 nginx performance: 0
2024-06-19 15:27:42 nginx performance: 0
2024-06-19 15:27:43 nginx performance: 0
2024-06-19 15:27:44 nginx performance: 0
2024-06-19 15:27:45 nginx performance: 0
2024-06-19 15:27:46 nginx performance: 0
```

2）客户端：

./test/nginx_ab_client.sh

参数 1：server IP 地址

参数 2：进程个数

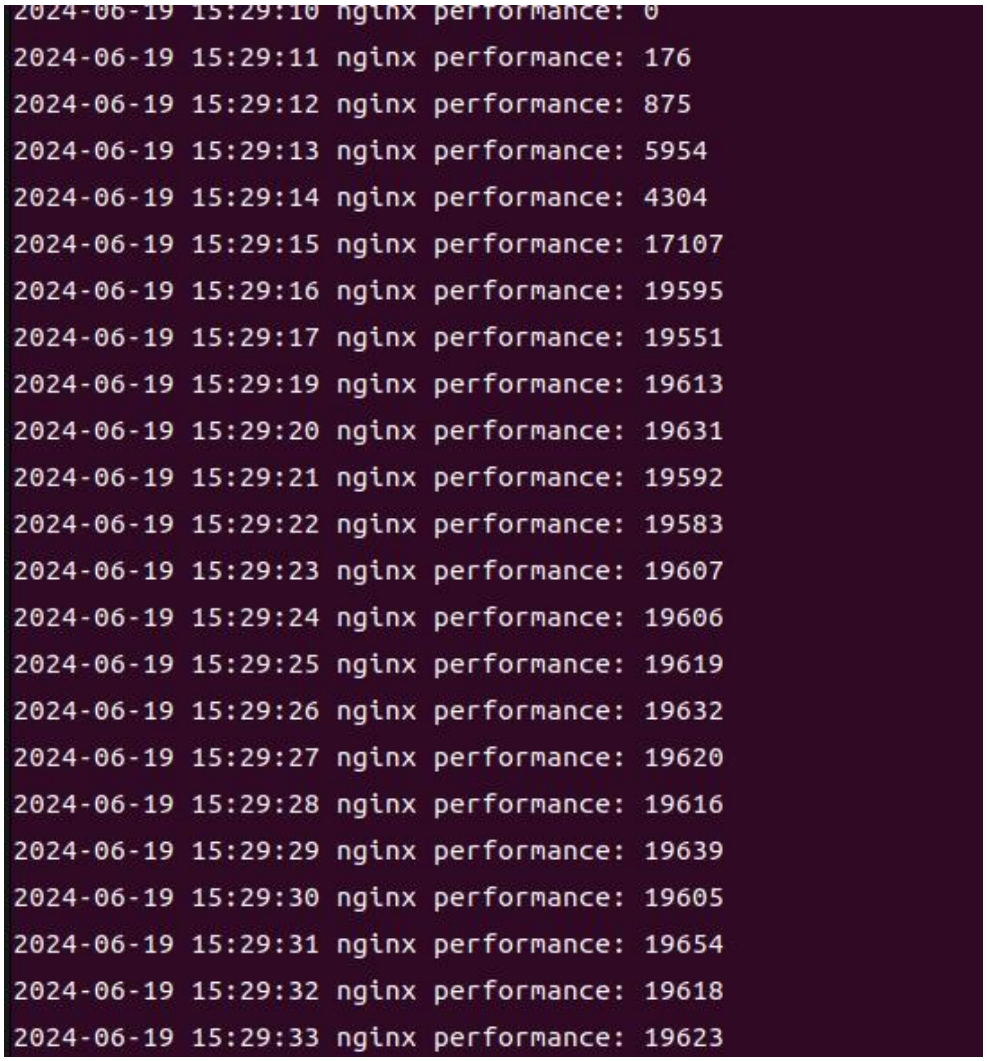
参数 3：绑定计算核心起始 ID

参数 4：SSL 引擎模式，支持 openssl 和 gmssl

下图是客户端启动输出

```
[root@localhost driver_3755_02_ssl_engine_1.0.0.0]# ./test/nginx_ab_client.sh localhost 8 8 openssl

Run AB Test, Process number: 8, Multiple requests number: 16, Core number:

2024-06-19 15:29:12 ab process cnt: 16
2024-06-19 15:29:13 ab process cnt: 16
2024-06-19 15:29:14 ab process cnt: 16
2024-06-19 15:29:15 ab process cnt: 16
2024-06-19 15:29:16 ab process cnt: 16
2024-06-19 15:29:17 ab process cnt: 16
2024-06-19 15:29:18 ab process cnt: 16
2024-06-19 15:29:19 ab process cnt: 16
2024-06-19 15:29:20 ab process cnt: 16
2024-06-19 15:29:21 ab process cnt: 16
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
Completed 10000 requests
```

此时切换到服务端控制台，可以看到请求处理的性能，单位是每秒处理请求个数。



4.6. Docker 虚拟化测试

#测试前的准备
再物理机上安装 Docker，详见各 linux 版本说明

#测试前的准备
在物理机上， 修改 load_driver.sh
VF_N=0
VF_Q=0
PF_Q=64
然后运行 ./load_driver.sh

#创建 docker container
docker run --net=host --privileged -v /lib/modules:/lib/modules -v

```
/dev:/dev -v
/root/driver_3755_02_v2.0.3.7:/root/driver_3755_02_v2.0.3.7 --device
/dev/rsp-drv --device /dev/rsp-cmdq-drv  -d --name test03
centos:centos7  sleep infinity
```

#进入 docker container 执行配置和测试

```
docker exec -it test01 bash

cd /root/driver_3755_02_v2.0.3.7
./kmt  #密钥管理
./test #性能测试
```

4.7. KVM 虚拟化测试

#测试前的准备

1）首先确认 CPU 支持虚拟化 VT-d (Intel 处理器)或者 AMD-Vi (AMD 处理器)，并确保 VT-d/AMD-Vi 已经在 BIOS 中开启。

2）然后再 linux kernel 的启动命令行中增加 “intel_iommu=on”或“amd_iommu=on” 和 iommu=pt

配置启动命令行，可以参考如下步骤：

#vi /etc/default/grub

GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt"

然后运行

update-grub

后，重启机器

运行 “"dmesg | grep -e DMAR -e IOMMU"”, 如果有类似如下输出，可以确认配置正确

```
[    1.726112] pci 0000:00:00.2: AMD-Vi: IOMMU performance counters
supported
[    1.728781] pci 0000:00:00.2: AMD-Vi: Found IOMMU cap 0x40
[    1.729335] perf/amd_iommu: Detected AMD IOMMU #0 (2 banks, 4
counters/bank).
```

#修改 load_driver.sh 里面的参数，设置 VF 个数，PF，VF 的队列数：

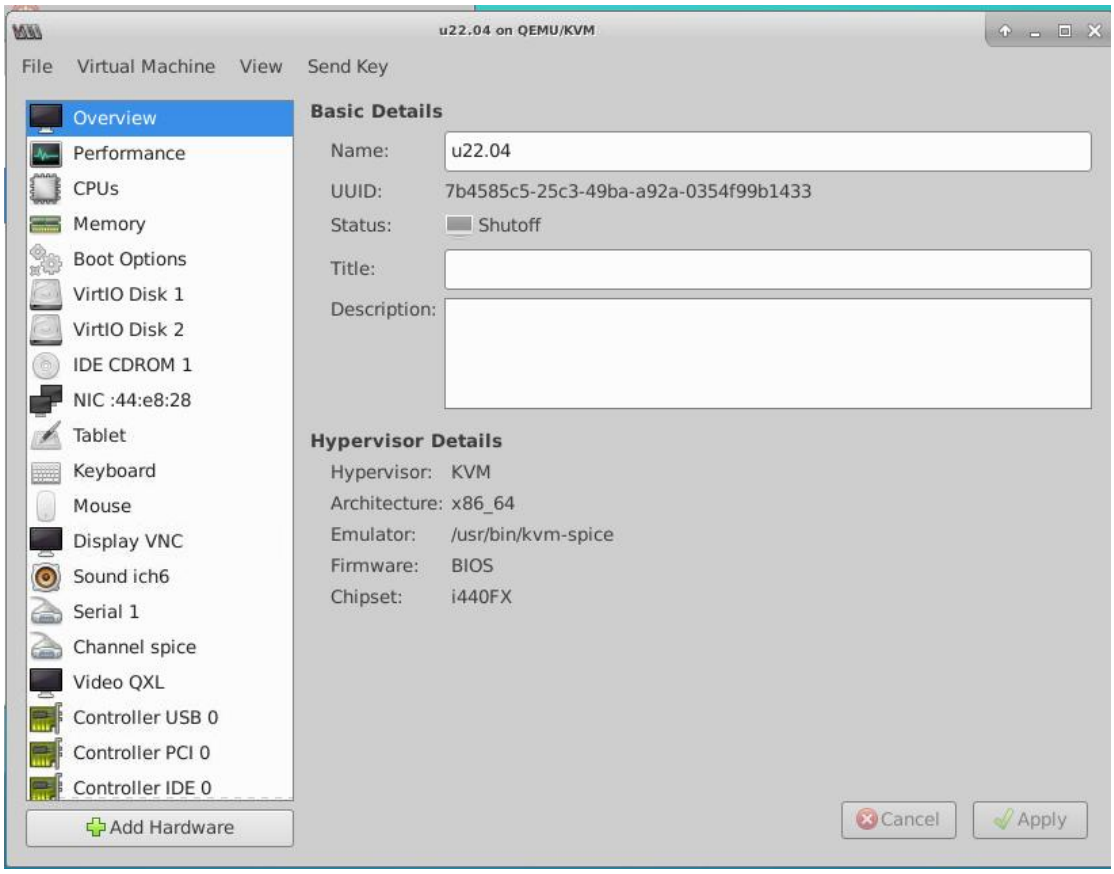
VF_N=4

VF_Q=4
PF_Q=4

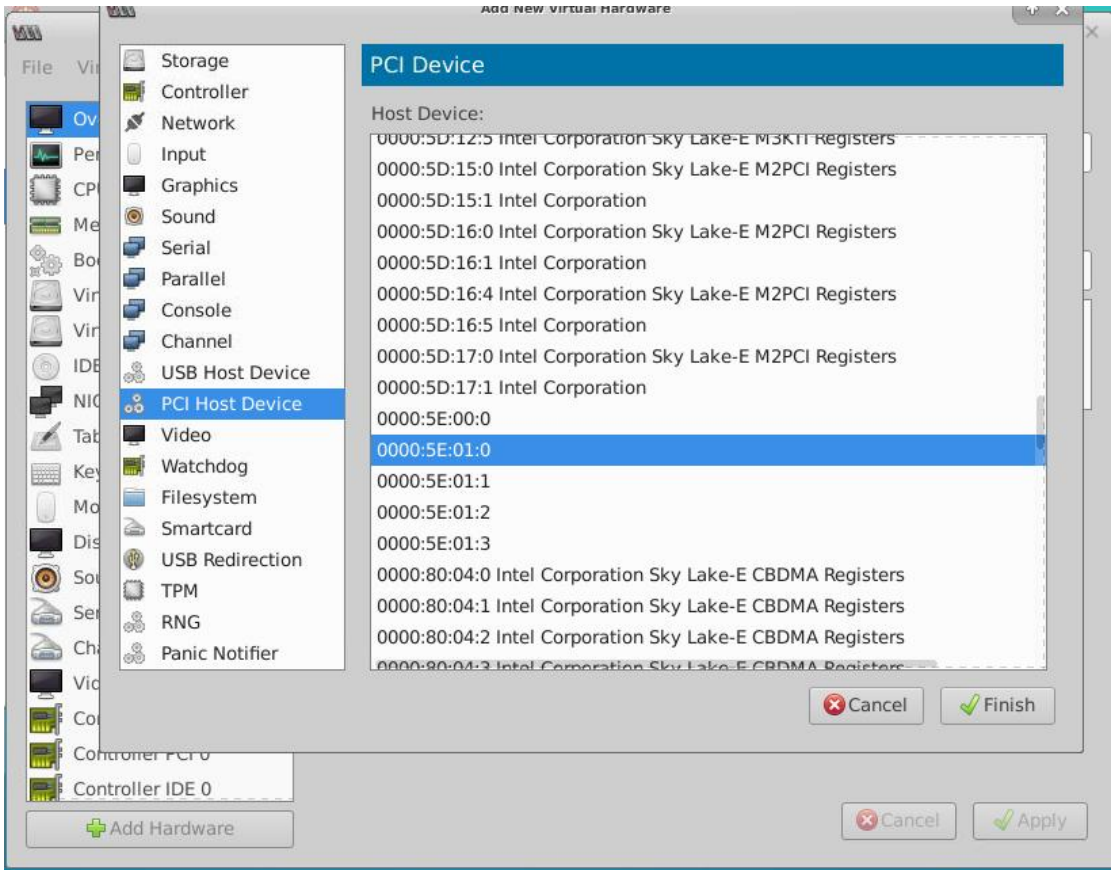
重启驱动
./unload_driver.sh; ./load_driver.sh

确认驱动已加载
lsmod | grep rsp

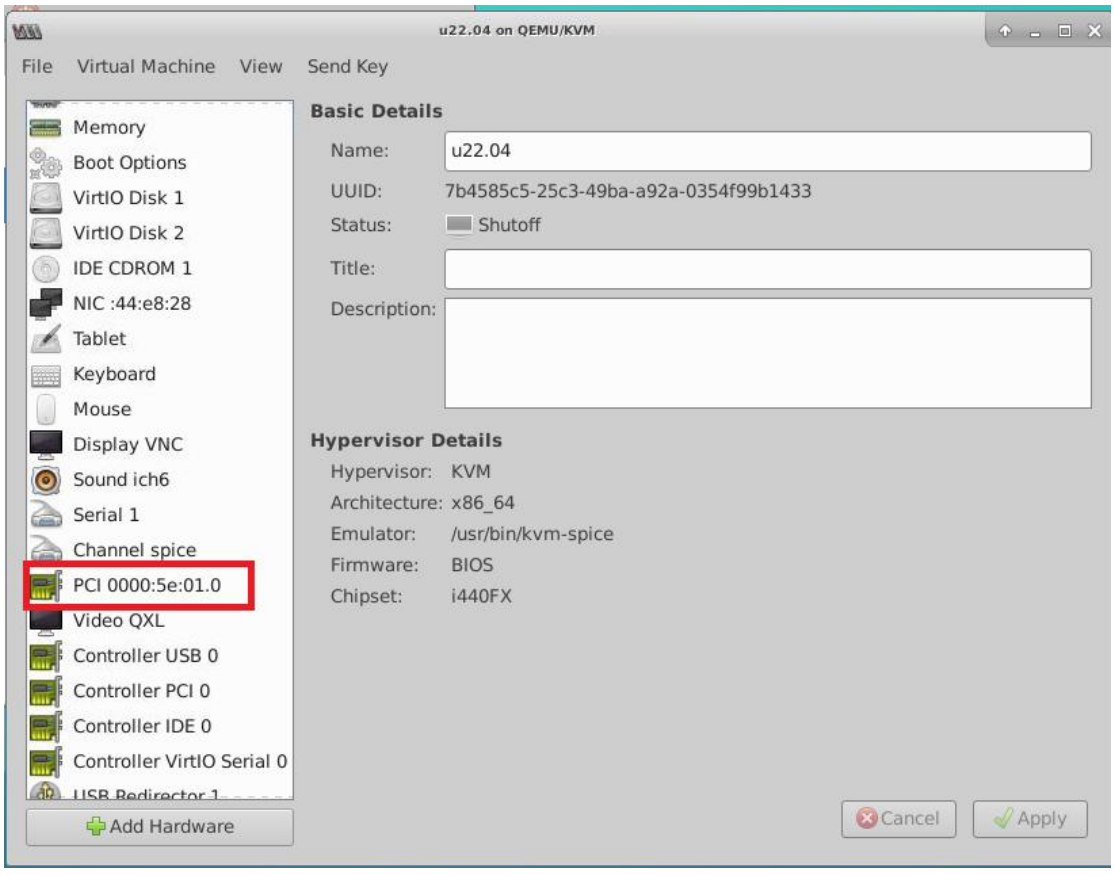
配置虚拟机（下面以 KVM 为例），Add Hardware



左边选择 PCI Host Device， 右边按设备 PCIe ClassID 向下查找到 VF 设备


Add New Virtual Hardware
Storage
Controller
Network
Input
Graphics
Sound
Serial
Parallel
Console
Channel
USB Host Device
PCI Host Device
Video
Watchdog
Filesystem
Smartcard
USB Redirection
TPM
RNG
Panic Notifier
PCI Device
Host Device:
0000:5D:15:0 Intel Corporation Sky Lake-E M2PCI Registers
0000:5D:15:1 Intel Corporation
0000:5D:16:0 Intel Corporation Sky Lake-E M2PCI Registers
0000:5D:16:1 Intel Corporation
0000:5D:16:4 Intel Corporation Sky Lake-E M2PCI Registers
0000:5D:16:5 Intel Corporation
0000:5D:17:0 Intel Corporation Sky Lake-E M2PCI Registers
0000:5D:17:1 Intel Corporation
0000:5E:00:0
0000:5E:01:0
0000:5E:01:1
0000:5E:01:2
0000:5E:01:3
0000:80:04:0 Intel Corporation Sky Lake-E CBDMA Registers
0000:80:04:1 Intel Corporation Sky Lake-E CBDMA Registers
0000:80:04:2 Intel Corporation Sky Lake-E CBDMA Registers
Cancel
Finish
Controller IDE 0
Add Hardware
Cancel
Apply

添加成功


u22.04 on QEMU/KVM
File
Virtual Machine
View
Send Key
Memory
Boot Options
VirtIO Disk 1
VirtIO Disk 2
IDE CDROM 1
NIC :44:e8:28
Tablet
Keyboard
Mouse
Display VNC
Sound ich6
Serial 1
Channel spice
PCI 0000:5e:01.0
Video QXL
Controller USB 0
Controller PCI 0
Controller IDE 0
Controller VirtIO Serial 0
Add Hardware
Basic Details
Name: u22.04
UUID: 7b4585c5-25c3-49ba-a92a-0354f99b1433
Status: Shutoff
Title:
Description:
Hypervisor Details
Hypervisor: KVM
Architecture: x86_64
Emulator: /usr/bin/kvm-spice
Firmware: BIOS
Chipset: i440FX
Cancel
Apply

登录进入虚拟机：

验证设备被识别

```
root@u2004test1:~# lspci | grep 1dab
00:0c.0 Encryption controller: Device 1dab:8001 (rev 01)
root@u2004test1:~#
```

然后同物理机一样，在虚拟机内部编译，安装驱动，进行测试。

4.8. 多卡测试

同一个物理机可以插入多张物理卡以提高处理性能。

这是两个卡

```
root@serica-UIS-Cell-3030-G3:~# lspci -d 1dab:7002
18:00.0 Encryption controller: Device 1dab:7002 (rev 01)
5e:00.0 Encryption controller: Device 1dab:7002 (rev 01)
```

多卡进行性能测试时，运行如下命令
./run_testcase/run_benchmark.sh <dev_id(1~n)> <process/thread cnt> <sync(1:sync, 0:async)>

例如：
./run_testcase/run_benchmark.sh 1 8 0
./run_testcase/run_benchmark.sh 2 8 0

在独立的 console 里面同时运行上述命令，可以获得多卡同时运行时的性能。

多张物理卡可以分别虚拟化出 VF 设备， 下面是用命令配置 VF 的个数的例子。

```
root@serica-UIS-Cell-3030-G3:~# echo 8 >  /sys/bus/pci/devices/0000\:5e\:00.0/sriov_numvfs
root@serica-UIS-Cell-3030-G3:~# lspci | grep  1dab
18:00.0 Encryption controller: Device 1dab:7002 (rev 01)
5e:00.0 Encryption controller: Device 1dab:7002 (rev 01)
5e:01.0 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.1 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.2 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.3 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.4 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.5 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.6 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.7 Encryption controller: Device 1dab:8001 (rev 01)
```

第二张卡

```
root@serica-UIS-Cell-3030-G3:~# echo 4 >  /sys/bus/pci/devices/0000\:18\:00.0/sriov_numvfs
root@serica-UIS-Cell-3030-G3:~# lspci | grep  1dab
18:00.0 Encryption controller: Device 1dab:7002 (rev 01)
18:01.0 Encryption controller: Device 1dab:8001 (rev 01)
18:01.1 Encryption controller: Device 1dab:8001 (rev 01)
18:01.2 Encryption controller: Device 1dab:8001 (rev 01)
18:01.3 Encryption controller: Device 1dab:8001 (rev 01)
5e:00.0 Encryption controller: Device 1dab:7002 (rev 01)
5e:01.0 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.1 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.2 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.3 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.4 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.5 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.6 Encryption controller: Device 1dab:8001 (rev 01)
5e:01.7 Encryption controller: Device 1dab:8001 (rev 01)
```

5. SDK 编程接口概述

5.1. 国密 SDF 标准接口

SDF 接口支持 GM0018 规范的全部接口，方便用户应用对接。

下面是一个使用 SDF 接口的例子，关于接口 api 的详情，请参考软件包里面的 sdf.h 文件

```
1. #include "sdf.h"
2.
3. int main(int argc, char *argv[])
4. {
```

```
5.  SGD_HANDLE hDeviceHandle;
6.  SGD_HANDLE hSessionHandle;
7.  SGD_HANDLE hKeyHandle;
8.
9.  // 打开设备
10. result = SDF_OpenDevice(&hDeviceHandle);
11. if (result != SDR_OK)
12. {
13.  printf("SDF_OpenDevice failed\n");
14.  goto err;
15. }
16. // 创建会话
17. result = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
18. if (result != SDR_OK)
19. {
20.  printf("SDF_OpenSession failed\n");
21.  return CRYPTO_FAILURE;
22. }
23.
24. result = SDF_ImportKey(hSessionHandle, sm4_block_key, 16, &hKeyHandle);
25. if (result != SDR_OK)
26. {
27.  printf("打导入明文会话密钥错误 0x%x\n", result);
28. }
29.
30. result = SDF_Encrypt(hSessionHandle, hKeyHandle, SGD_SM4_ECB, NULL, inData, data_l
    en, encData, &encLen);
31.
32. if (result != SDR_OK || encLen != data_len)
33. {
34.  printf("%s failed\n", tname);
35.  goto err;
36. }
37.
38. err:
39. SDF_DestroyKey(hSessionHandle, hKeyHandle);
40. if (hSessionHandle)
41.  SDF_CloseSession(hSessionHandle);
42. if (hDeviceHandle)
43.  SDF_CloseDevice(hDeviceHandle);
44.
45. return CRYPTO_SUCCESS;
46. }
47.
```

5.2. SDFe 增强接口

SDFe 接口是对 SDF 接口的扩展，包含了加密卡提供的补充功能的使用接口。下面列出部分函数，全部函数详见 sdfe.h 文件

```
/* 扩展类设备管理相关函数 */

/*
 * [函数名称] SDFE_GetDeviceNumber
 * [功能描述] 获取设备个数
 * [参数描述] number: 输出设备个数
 * [ 返回值 ] 见 SDR 错误码
 */
int SDFE_GetDeviceNumber(uint32_t *number);

/*
 * [函数名称] SDFE_GetDeviceBDF
 * [功能描述] 获取设备 pcie 标识符
 * [参数描述] adapterid:输入设备 ID, 范围 0~32;
 *          pcie_bdf: pcie 标识符
 * [ 返回值 ] 见 cap_get_dev_bdf
 */
int SDFE_GetDeviceBDF(uint32_t adapterid, int *pcie_bdf);

/*
 * [函数名称] SDFE_OpenDevice
 * [功能描述] 打开密码卡设备, 该接口支持指定密码卡 ID.
 * [参数描述] adapterid:    输入设备 ID, 范围 0~32;
 *          phDeviceHandle: 输出已打开的设备句柄
 * [ 返回值 ] 见 SDR 错误码
 */
int SDFE_OpenDevice(uint32_t adapterid, void** phDeviceHandle);

/*
 * [函数名称] SDFE_GetSecurityLevel
 * [功能描述] 获取密码卡安全等级
 * [参数描述] hSessionHandle: 输入会话句柄
 *          plevel:    输出安全等级, 见 SDFE_SecurityLevel 定义;
 * [ 返回值 ] 见 SDR 错误码
 */
int SDFE_GetSecurityLevel(void* hSessionHandle, uint32_t* plevel);
```

。。。

5.3. CAP 内部编程接口

CAP 内部接口是功能更丰富，效率更高的接口，包含了算法 API 和密钥管理 API，支撑上层国密 SDF 标准接口和 SDFe 增强接口，并且支持同步调用和异步调用方式。
下面是同步调用的例子：

```
1.      ...
2.    cap_open_device("rsp_dev_01", &hdev, POLL_WAIT_US);
3.
4.    cap_cipher_ctx_t cap_cipher;
5.    memset(&cap_cipher, 0, sizeof(cap_cipher_ctx_t));
6.
7.    cap_cipher.hdev = hdev;
8.    cap_cipher.sess.mode = CAP_SYNC_MODE;
9.
10.   while(CAP_RET_BUSY == cap_cipher_onetime(&cap_cipher, CAP_CIPHER_SM4,CAP_CI
    PHER_ECB, CAP_CIPHER_ENCRYPT, msg, sizeof(msg),
11. hw_out, &outlen, key, 32, iv, 16, NULL, 0))
12.   {
13.     usleep(SEND_PKG_WAIT_US);
14.   }
15.
16.   cap_close_device(hdev);
17.     ...
18.
```

下面是查询方式异步调用的例子：

```
1.      ...
2.    cap_open_device("rsp_dev_01", &hdev, POLL_WAIT_US);
3.
4.    cap_cipher_ctx_t cap_cipher;
5.    memset(&cap_cipher, 0, sizeof(cap_cipher_ctx_t));
6.
7.    cap_cipher.hdev = hdev;
8.    cap_cipher.sess.mode = CAP_ASYNC_POLLING_MODE;
9.
10.   while(CAP_RET_BUSY == cap_cipher_onetime(&cap_cipher, CAP_CIPHER_SM4,CAP_CI
    PHER_ECB, CAP_CIPHER_ENCRYPT, msg, sizeof(msg),
11. hw_out, &outlen, key, 32, iv, 16, NULL, 0))
12.   {
13.     usleep(SEND_PKG_WAIT_US);
```

```
}
  ...做一些其他操作，或者发送另一个加密请求...
while(cap_cipher.sess.state == CAP_SESS_STATE_BUSY) usleep(SEND_PKG_WAIT_US);

cap_close_device(hdev);
  ...

```

下面是回调方式异步调用的例子：

```
...
void cap_cipher_cb(void *cb_param)
{
    cap_cipher_ctx_t *cap_cipher = (cap_cipher_ctx_t *)cb_param;
    ...
}
...
cap_open_device("rsp_dev_01", &hdev, POLL_WAIT_US);

cap_cipher_ctx_t cap_cipher;
memset(&cap_cipher, 0, sizeof(cap_cipher_ctx_t));

cap_cipher.hdev = hdev;
cap_cipher.sess.mode = CAP_ASYNC_CALLBACK_MODE;
cap_cipher.sess.cb = cap_cipher_cb;
cap_cipher.sess.cb_param = &cap_cipher;

 while(CAP_RET_BUSY == cap_cipher_onetime(&cap_cipher, CAP_CIPHER_SM4,CAP_CI
PHER_ECB, CAP_CIPHER_ENCRYPT, msg, sizeof(msg),
hw_out, &outlen, key, 32, iv, 16, NULL, 0))
{
    usleep(SEND_PKG_WAIT_US);
}

cap_close_device(hdev);
  ...

```

CAP 所有接口详见 sdk/cap/cap.h 文件。

6. 密钥管理概述

6.1. 工具简介

密码卡管理工具可用于完成密码卡的初始化配置，包括设备管理、用户管理、密钥管理和文件管理功能；

密码卡支持三种访问控制策略：主机端 Ukey 访问控制模式、口令访问控制模式、无访问控制模式。

当配置为主机端 Ukey 访问控制模式时，支持三个管理员 Ukey 和 1 个操作员 Ukey。密码卡出厂默认配置为无访问控制模式。

后面是基本功能操作的例子，其他操作类似。

6.2. 查询设备状态

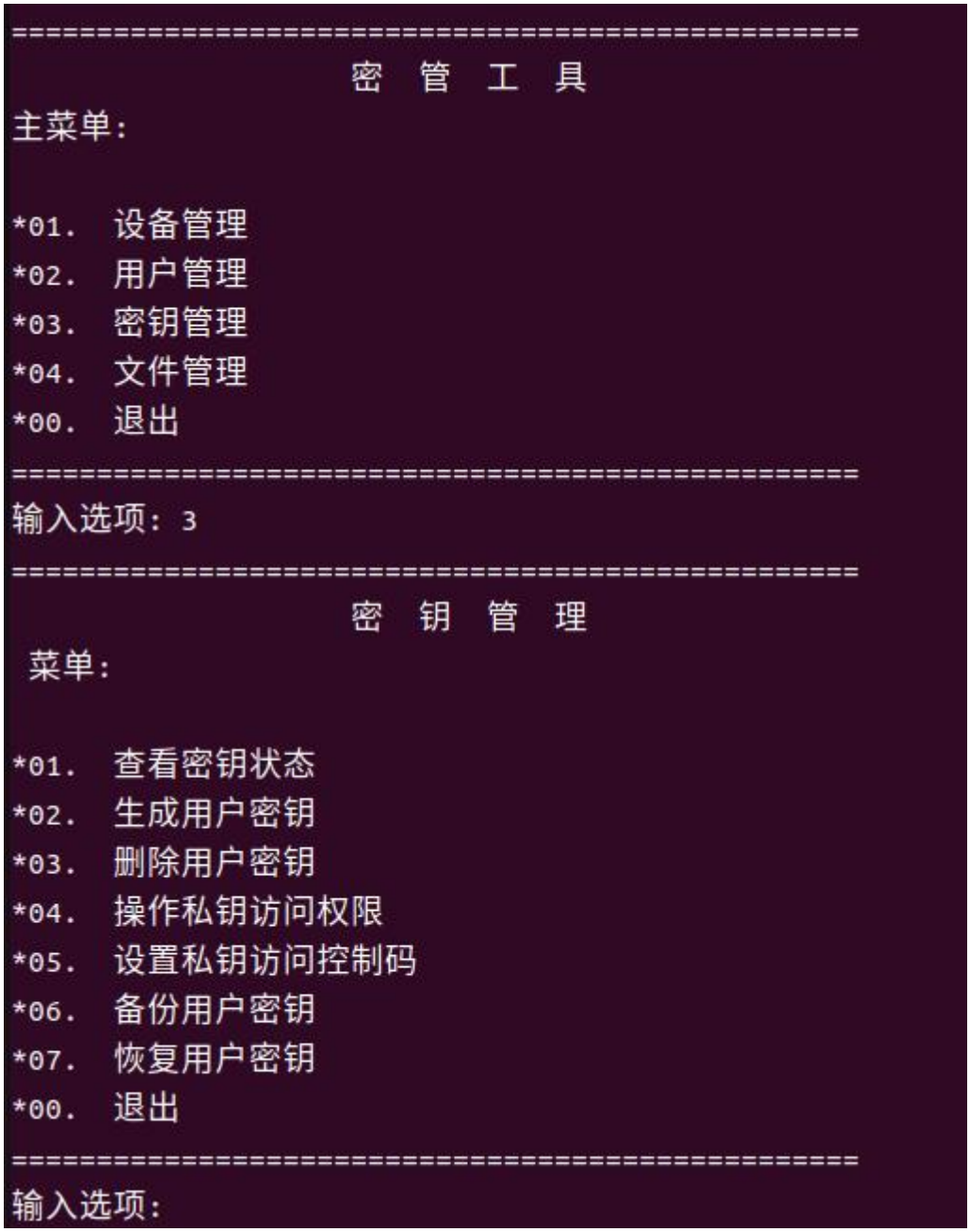
启动命令行密钥管理工具如下图

顺序输入 1 ， 1 查询设备信息

```
*01.  设备管理
*02.  用户管理
*03.  密钥管理
*04.  文件管理
*00.  退出
==================================================
输入选项: 1
==================================================
                    设 备 管 理
 菜单:

*01.  获取设备信息
*02.  获取设备状态
*03.  获取设备配置
*04.  设备初始化
*05.  设置设备序列号
*06.  设备自检
*07.  固件升级
*08.  恢复出厂
*09.  固件日志等级
*00.  退出
==================================================
输入选项: 1

厂商名称: Serica
设备名称: Gemini 3755
设备序列号: e4b129f7f900d606
接口版本: 0x02000306
固件版本: 2.0.3.6

支持的非对称算法: RSA_SIGN|RSA_ENC|SM2_1|SM2_2|SM2_3
获取设备信息成功!
```

6.3. 查询密钥信息

依次输入 3，1 进入密钥信息查询：



输入想要查询的密钥类型：

```
请输入密钥类型:
        1 - SM2 签名密钥对
        2 - SM2 加密密钥对
        3 - RSA 签名密钥对
        4 - RSA 加密密钥对
        5 - 密钥加密密钥
        6 - 会话密钥
        0 - 退出
:
```

按 1，查询出的 SM2 签名密钥对如下：

```
SM2 签名密钥 [密钥索引 - 245] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 246] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 247] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 248] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 249] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 250] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 251] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 252] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 253] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 254] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 255] 状态 : 空闲, 长度 0位
SM2 签名密钥 [密钥索引 - 256] 状态 : 空闲, 长度 0位
查看密钥状态成功!
```

按 5，查询出的对称加密密钥对如下：

```
密钥保护密钥 [密钥索引 - 1013] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1014] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1015] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1016] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1017] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1018] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1019] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1020] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1021] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1022] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1023] 状态 : 空闲, 长度 0位
密钥保护密钥 [密钥索引 - 1024] 状态 : 空闲, 长度 0位
查看密钥状态成功!
```