AG35-QuecOpen

TZ 加密应用指导

LTE 系列

版本：EC2X&AG35-QuecOpen_TZ 加密_使用指导_V1.0

日期：2017-12-04

状态：临时文件

上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市徐汇区虹梅路 1801 号宏业大厦 7 楼　　邮编：200233
电话：+86 21 51086236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：
http://quectel.com/cn/support/sales.htm

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：
http://quectel.com/cn/support/technical.htm
或发送邮件至：support@quectel.com

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。由于客户操作不当而造成的人身伤害或财产损失，本公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

版权申明

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2017-12-04	钱润生	初始版本

目录

表格索引

图片索引

1 QSEE/TZ 加密介绍

1.1. TZ 加密服务

在 ARM TZ 架构下，物理层面非安全世界的 HLOS 系统通过 SMC(secure monitor call)接口访问安全世界 TZ，软件层面通过 Linux 系统的 SCM（secure channel manager）框架接口进行通信交互。
MDM9628 TZ CRYPTO 提供如下三种功能：
（1）随机数生成
随机数长度 1-512 个字节
（2）对称密钥生成，密钥保护，数据加解密
仅支持 AES256-CCM
（3）非对称密钥生成，密钥保护，数据签名与验签
仅支持 RSA 算法，密钥长度 1024 或者 2048。签名摘要算法仅支持 SHA256，编码格式支持 PKCS#1v1.5 或者 PSS

备注

目前平台不支持**安全存储**和**密钥存储**。

1.2. Quectel TZ 加密实现

1.2.1. 系统调用

Quectel 在 Linux 系统上基于 scm call 实现访问高通 TZ CRPTO 的内核驱动，用户只需通过打开/dev/tzone 设备，通过 ioctl 系统调用实现各种加密服务。参照 ql_tzone.h，可以知道实现服务 cmd：

QL_TZ_CRYPTO_SVC_RANDOM_DATA_GEN
QL_TZ_CRYPTO_SVC_AES_KEY_GEN
QL_TZ_CRYPTO_SVC_AES_KEY_IMPORT
QL_TZ_CRYPTO_SVC_AES_DATA_ENCRYPT
QL_TZ_CRYPTO_SVC_AES_DATA_DECRYPT
QL_TZ_CRYPTO_SVC_RSA_KEYPAIR_GEN
QL_TZ_CRYPTO_SVC_RSA_DATA_SIGN
QL_TZ_CRYPTO_SVC_RSA_SINDATA_VERIFY
QL_TZ_CRYPTO_SVC_RSA_KEYPAIR_IMPORT
QL_TZ_CRYPTO_SVC_RSA_PUBKEY_EXPORT

关于各个服务的使用请参照 ql_tzapp.c 文件。

1.2.2. Openssl-tz

Quectel 基于 openssl engine 技术，实现了基于 openssl API 实现调用 tz 加密服务，可直接参靠 example/crypto-tz 目录下面的 example。

2 随机数

请求随机数，只需要注意请求随机数长度，单位是 BYTE.合法区间值 1-512.
主要步骤：

```
….
ql_tz_random_data_t random_data = {512, NULL};
random_data.data_ptr = malloc(random_data.data_size);
ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RANDOM_DATA_GEN, &random_data);
….
```

3 AES 服务

3.1. 密钥生成

TZ 生成密钥有两种方式：硬件随机数产生和用户导入，最后提供给非安全世界一个加密的密钥块（blob），keyblob 由：Encryptedkey(256bit)+Nonce(64bit)+Authtaglen(128bit)组成。
限制：密钥长度 256bit

3.1.1. 基于随机数

关键步骤：

```
ql_tz_aes_generate_key_t keyblob = {256, NULL};
keyblob.key_ptr = malloc(keyblob.key_size);
ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_AES_KEY_GEN, &keyblob);
```

3.1.2. 基于用户导入

关键步骤：

```
ql_tz_aes_import_key_t ikeyblob;
    uint8 g_AESKey[] = {
        0x00,0xd5,0x6e,0x89,0x6a,0xdb,0x76,0xe5, 0xe4,0xf6,0x2d,0xdd,0xa3,0xaf,0x1b,0x15,
        0x00,0xd5,0x6e,0x89,0x6a,0xdb,0x76,0xe5, 0xe4,0xf6,0x2d,0xdd,0xa3,0xaf,0x1b,0x15
};//任意数据，必须是 32 个字节（256bit）

ikeyblob.input_aes_size = sizeof(g_AESKey);
ikeyblob.input_aes_ptr = g_AESKey;
    ikeyblob.key_size = 56;//大于等于 56
    ikeyblob.key_ptr = malloc(ikeyblob.key_size);

ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_AES_KEY_IMPORT, &ikeyblob);
```

3.2. 数据加密

限制：明文长度 2048 字节
关键步骤：

```
//读入密钥和明文数据，存放到 keyblob 及 input_data_ptr
…..
    enc_data.key_size = keyblob.key_size;
    enc_data.key_ptr = keyblob.key_ptr;
    enc_data.input_data_size = input_data_size;
    enc_data.input_data_ptr = input_data_ptr;
    enc_data.output_data_size = output_data_size;
```

```
enc_data.output_data_ptr = output_data_ptr;

ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_AES_DATA_ENCRYPT, &enc_data);
```

3.3. 数据解密

限制：密文长度 2048 字节

关键步骤：

```
    //读入密钥和密文，存放到 keyblob 和 input_data_ptr
    ……
dec_data.key_size = keyblob.key_size;
    dec_data.key_ptr = keyblob.key_ptr;
    dec_data.input_data_size = input_data_size;
    dec_data.input_data_ptr = input_data_ptr;
    dec_data.output_data_size = output_data_size;
    dec_data.output_data_ptr = output_data_ptr;

ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_AES_DATA_DECRYPT, &dec_data);
```

4 RSA 服务

4.1. 密钥生成

TZ 生成非对称密钥有两种方式：硬件随机数产生和用户导入，最后提供给非安全世界一个加密的密钥块（blob），keyblob 由：Encrypted PrivateBlob+Public key+HMAC 组成。

限制：modulus 大小可以是 1024、2048，公钥指数 exponet 为：65537；签名摘要算法 SHA256,编码 PKCS#1V1.5 或 PSS，签名长度 256 个字节。如果 moduls 大小为 2048，数据返回时间大约需要 3s

4.1.1. 基于随机数

关键步骤：

```
ql_tz_rsa_generate_key_t rsa_key_blob_req;

rsa_key_blob_req.modulus_size = 2048;
    rsa_key_blob_req.public_exponent = 0x010001;//65537
rsa_key_blob_req.digest_pad_type = QL_TZ_RSA_PKCS115_SHA2_256;

rsa_key_blob_req.rsa_key_blob = malloc(sizeof(ql_crypto_rsa_key_blob_t));

ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RSA_KEYPAIR_GEN, &rsa_key_blob_req);
```

4.1.2. 基于用户导入

关键步骤

```
    //用户可以使用 openssl genrsa 命令生成密钥，然后用 openssl rsa 将密钥信息打印出来
    ql_tz_rsa_keypair_import_t import_keypair;
    ql_crypto_rsa_key_blob_t   tz_key_blob;

    import_keypair.in_modulus = g_modulus;
    import_keypair.in_modulusSize = sizeof(g_modulus);
    import_keypair.in_pubExp = g_publicExponent;
    import_keypair.in_pubExpSize = sizeof(g_publicExponent);
    import_keypair.in_privExp = g_privateExponent;
    import_keypair.in_privExpSize = sizeof(g_privateExponent);
    import_keypair.in_padding = QL_TZ_RSA_PKCS115_SHA2_256;
    import_keypair.rsa_key_blob = malloc(sizeof(ql_crypto_rsa_key_blob_t));

    ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RSA_KEYPAIR_IMPORT, &import_keypair);
```

4.2. 数据签名

限制：明文数据长度 2048
关键步骤：

```
ql_tz_rsa_sign_verify_data_t rsa_sign_data;

signature_length = 256;
    signature_ptr = malloc(signature_length);

    rsa_sign_data.input_data_size = input_data_size;
    rsa_sign_data.input_data_ptr = input_data_ptr;
    rsa_sign_data.signature_length = signature_length;
    rsa_sign_data.signature_ptr = signature_ptr;

    ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RSA_DATA_SIGN, &rsa_sign_data);
```

4.3. 数据验签

关键步骤：

```
  ql_tz_rsa_sign_verify_data_t verify_data;

    verify_data.input_data_size = input_data_size;
    verify_data.input_data_ptr = input_data_ptr;
    verify_data.signature_length = signature_length;
    verify_data.signature_ptr = signature_ptr;

    ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RSA_SINDATA_VERIFY, &verify_data);
```

4.4. 公钥导出

关键步骤：

```
    export_pubkey.modulus_data_size = 2048;//>256
    export_pubkey.modulus_data_ptr = malloc(export_pubkey.modulus_data_size);
    export_pubkey.public_exponent_size = 32;//>=8
    export_pubkey.public_exponent_ptr = malloc(export_pubkey.public_exponent_size);

ret = ioctl(tz_fd, QL_TZ_CRYPTO_SVC_RSA_PUBKEY_EXPORT, &export_pubkey);
```