使用 Python 来操作 Microchip 安全芯片

Microchip 提供业界优秀的安全方案，包括支持 SHA256，ECC P256 和 AES128 的完整算法支持，同时提供了基于 C 语言的器件支持库 cryptoauthlib.

由于 Python 语言的通用性和便利性，Microchip 也提供了基于 Python 语言的 CryptoAuthLib 和相关的例程 cryptoauthtools。

CryptoAuthLib Python 库可以做什么 ？

CryptoAuthLib 库提供了访问安全器件大部分功能的模块，这些模块只是对 C 语言库 cryptoauth 中的 API 做了一层封装，以实现在 Python 中的调用。

Microchip 的 cryptoauthlib 的访问页面如下: CryptoAuthLib Link

准备工作

软件部分

首先从 github 上下载最新的例程代码，地址如下： https://github.com/MicrochipTech/cryptoauthtools

安装 Python，最好是 3.x 的版本。如果已经安装过可以忽略这一步。

安装 Python 需要的组件。 进入 cryptoauthtools\python\examples 目录中，运行以下命令:

```
pip install -r requirements.txt
```

Python 将自动安装 cryptoauthlib 和 cryptography 库(需要 v2.3 以上的版本)

另外，也可以手动安装 cryptoauthlib 和 cryptography

```
pip install cryptoauthlib
pip install cryptography
```

硬件部分

支持的硬件如下:

- AT88CK101
- CryptoAuthentication Starter Kit (DM320109)
- ATECC508A, ATECC608A, ATSHA204A device directly connected via I2C (Linux Only)

支持的器件型号如下:

- ATSHA204A
- ATECC508A
- ATECC608A

下面的演示是在 Windows 10 64bit，Python 3.6 下运行的，使用的硬件是 CryptoAuthentication Starter Kit。这个开发套件实际上是使用 SAMD21 Xplained Pro + IC 转接座组成的。用 SAMD21 实现 USB HID 通信，再转换为 I2C 接口控制安全芯片。

- Tips: 需注意 USB 线应该插到 Target USB 上。

可以自行将 SAMD21 的开发板烧录一个固件来实现。这个固件可以在官网下载 ATCRYPTOAUTHSSH-XSTK_v1.0.1.zip

运行例程

这些例子旨在简单明了地说明基本概念。 要获得任何示例的帮助，您可以参考相关文档(例如:info.py 有一个 info.md 说明文档)或者通过命令行:

```
  $ info.py -h
usage: info.py [-h] [-i {i2c,hid}] [-d {ecc,sha}] [-p [PARAMS [PARAMS ...]]]
```

示例列表：

- **info.py**: 读器件信息. 详见 **info.md**
- **config.py**: 配置器件. 详见 **config.md**
- **ecdh.py**: 演示 ECDH 运算. 详见 **ecdh.md**
- **sign_verify.py**: 演示 ECDSA 签名和验证. 详见 **sign_verify.md**
- **read_write.py**: 演示从 Slot 中连续加密写和读. 详见 **read_write.md**

Info 例程

这个示例从设备中提取标识信息和配置。

- 设备类型识别和掩码 OTP 修改
- 序列号
- 配置区数据
- 锁状态
- 器件公钥(Slot0 中存放器件私钥)

可选的参数：

```
-h, --help            显示帮助信息
-i {i2c,hid}, --iface {i2c,hid}
                      接口类型 (默认: hid)
-d {ecc,sha}, --device {ecc,sha} 器件类型(默认: ecc)
-p [PARAMS [PARAMS ...]], --params [PARAMS [PARAMS ...]]
  接口参数如 key=value
```

如果使用 ATSHA204A，需要在参数中增加 -d sha。ATECC508A 和 ATECC608 则可以不用加。

在使用 SHA204A 时，原代码中导出公钥会出错。需要增加一个条件判断，如下：

```
#Load the public key
   if 'ecc' == device and data_zone_locked:
       print('\nLoading Public key\n')
       public_key = bytearray(64)
       assert atcab_get_pubkey(0, public_key) == ATCA_SUCCESS
```

```
        public_key =
bytearray.fromhex('3059301306072A8648CE3D020106082A8648CE3D030107034200004') +
bytes(public_key)
        public_key = base64.b64encode(public_key).decode('ascii')
        public_key = ''.join(public_key[i:i+64] + '\n' for i in
range(0,len(public_key),64))
        public_key = '-----BEGIN PUBLIC KEY-----\n' + public_key + '-----END PUBLIC KEY-
----'
        print(public_key)
```

使用 CryptoAuthentication Starter Kit 和 CryptoAuth Xplained Pro 运行如下：

```
$ python info.py

Device Part:
    ATECC508A

Serial number:
    01 23 6B 3F AC 10 2F 1B A5

Configuration Zone:
    01 23 6B 3F 00 00 50 00 AC 10 2F 1B A5 00 45 00
    B0 00 55 00 8F 20 C4 44 87 20 87 20 8F 0F C4 36
    9F 0F 82 20 0F 0F C4 44 0F 0F 0F 0F 0F 0F 0F 0F
    0F 0F 0F 0F FF FF FF FF 00 00 00 00 FF FF FF FF
    00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF
    FF FF FF FF 00 00 00 00 FF FF 00 00 00 00 00 00
    33 00 1C 00 13 00 13 00 7C 00 1C 00 3C 00 33 00
    3C 00 3C 00 3C 00 30 00 3C 00 3C 00 3C 00 30 00

Check Device Locks
    Config Zone is locked
    Data Zone is locked

Loading Public key

-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE5O6vcWT7anlt+HFc6AAkF+rOBtly
uOXuM78qcnXtIA+nxvPzSWWiF0yruZK/4ANK8q2C21dICxDwxo7YyYpc4w==
-----END PUBLIC KEY-----
```

```
Done
```

Config 例程

在使用加密身份验证设备之前，必须为器件设置好适当的配置。 配置可能非常复杂，需要仔细创建和分析配置交互可能带来的安全漏洞。 开发新配置时，请同时查阅数据表和联系 FAE。

这个脚本将编写一个通用配置程序，允许对各种示例进行评估。 在生产设备使用之前，应该修改这个配置。 为了让实验人员更容易使用配置，下面列出了一些设置项。

ATECC508A 配置

- Slot 4 & 6 允许非加密写入，这意味着保护密钥可以在不需要很强安全性的前提下可以被改写。

```
# Example configuration for ATECC508A minus the first 16 bytes which are fixed by the factory
_atecc508_config = bytearray.fromhex(
    'B0 00 55 00 8F 20 C4 44 87 20 87 20 8F 0F C4 36'
    '9F 0F 82 20 0F 0F C4 44 0F 0F 0F 0F 0F 0F 0F 0F'
    '0F 0F 0F 0F FF FF FF FF 00 00 00 00 FF FF FF FF'
    '00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF'
    'FF FF FF FF 00 00 55 55 FF FF 00 00 00 00 00 00'
    '33 00 1C 00 13 00 13 00 7C 00 1C 00 3C 00 33 00'
    '3C 00 3C 00 3C 00 30 00 3C 00 3C 00 3C 00 30 00')
```

ATECC608A 配置:

- 同 ATECC508A

- Slot 4 是额外增加的，用作从 tempkey 中读取 ECDH 前导密钥和 KDF 因子的保护密钥

```
# Example configuration for ATECC608A minus the first 16 bytes which are fixed by the factory
_atecc608_config = bytearray.fromhex(
    'B0 00 55 01 8F 20 C4 44  87 20 87 20 8F 0F C4 36'
    '9F 0F 82 20 0F 0F C4 44  0F 0F 0F 0F 0F 0F 0F 0F'
    '0F 0F 0F 0F FF FF FF FF  00 00 00 00 FF FF FF FF'
    '00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00'
    '00 00 00 00 00 00 55 55  FF FF 06 40 00 00 00 00'
    '33 00 1C 00 13 00 13 00  7C 00 1C 00 3C 00 33 00'
    '3C 00 3C 00 3C 00 30 00  3C 00 3C 00 3C 00 30 00')
```

ATSHA204A 配置:

- Solt 可自由写入，无需事先知道 Slot 中的内容

```
_atsha204_config = bytearray.fromhex(
    'C8 00 55 00 8F 80 80 A1 82 E0 C4 F4 84 00 A0 85'
    '86 40 87 07 0F 00 C4 64 8A 7A 0B 8B 0C 4C DD 4D'
    'C2 42 AF 8F FF 00 FF 00 FF 00 FF 00 FF 00 FF 00'
    'FF 00 FF 00 FF FF FF FF FF FF FF FF FF FF FF FF'
    'FF FF FF FF 00 00 55 55')
```

上述示例主要是为了演示。实际使用中有一些安全性常识特别需注意的:

- 设备的用户/实验者必须仔细设置和管理密钥。
- 否则，如果这些密钥丢失或不小心设置，就会使设备变得异常。

如果设备之前没有被配置为可以使用，那么可以编写一个支持许多用例的基本配置(这是不可逆的)。

运行配置脚本可以如下运行：

查看脚本的可选项和帮助信息:

```
$ python config.py -h
```

运行配置脚本:

```
$ python config.py
```

ECDH 例子

ECDH 命令实现了椭圆曲线 Diffie-Hellman 算法，将内部私钥与外部公钥组合在一起，以计算共享密钥。 示例中 genkey 命令生成两个独立的 ECC 密钥对，然后通过 ECDH 命令计算共享密钥。

如果设备之前没有被配置为可以使用，那么可以编写一个支持许多用例的基本配置(这是不可逆的)。

```
$ python config.py
```

运行此示例的步骤: 查看脚本的可选项和帮助信息:

```
$ python ecdh.py -h
```

运行 ECDH 脚本:

```
$ python .\ecdh.py

Performing ECDH operations in the clear - see datasheet for encryption details

Host Public Key:
    9C 9C 1F 4D 71 0F 39 6E C0 49 80 E3 C7 A5 FA F7
    89 E6 39 F4 55 13 49 51 16 1D AA 93 58 52 07 7F
    87 36 AD DA 8E 4F 88 BF 51 29 F3 08 94 28 FC 7C
```

```
    C8 82 B5 33 AD E8 93 8B FC 52 33 35 22 F6 E2 3C

Device public key:
    49 E5 CE F0 48 F9 3C 81 D0 99 6D F0 E9 BD 6F 70
    A3 C9 7C 61 46 89 26 49 1E 0B E1 24 C1 42 95 58
    A3 8F 97 8B E6 CF 78 64 EC C8 81 7A A0 4D E8 BB
    E1 4E 99 F7 35 31 ED E1 05 88 97 37 BE B9 E1 2D

Host Calculated Shared Secret:
    E5 0B 40 8F F7 8F C7 92 4B AA 5F 04 9F 5E 16 64
    E7 80 11 A3 FC 7D B4 8E 17 4B 05 44 3E 85 12 2E

Device Calculated Shared Secret:
    E5 0B 40 8F F7 8F C7 92 4B AA 5F 04 9F 5E 16 64
    E7 80 11 A3 FC 7D B4 8E 17 4B 05 44 3E 85 12 2E

Comparing host and device generated secrets:
    Success - Generated secrets match!

Done
```

read_write 示例

Read 命令从设备的一个内存区域读取。数据在返回到系统之前可以选择加密。 写入命令写入设备上的 EEPROM 区域。 根据槽的 WriteConfig 字节的值，数据可能需要在发送到设备之前由系统加密 这个例子说明了明文写、明文读、加密读和加密写的使用

```
$ python .\read_write.py -d sha

Basic Read/Write Example

Generaing data using RAND command
    Generated data:
        31 38 23 06 13 74 7E 6B 3C 3B 10 76 01 D8 3A F3
        C3 27 3F 73 22 E4 7F 64 3C 89 D6 2A B9 FA 22 0D

Write command:
    Writing data to slot 8
```

```
    Write Success

Read command:
    Reading data stored in slot 8
    Read data:
        31 38 23 06 13 74 7E 6B 3C 3B 10 76 01 D8 3A F3
        C3 27 3F 73 22 E4 7F 64 3C 89 D6 2A B9 FA 22 0D

Verifing read data matches written data:
    Data Matches!

Writing IO Protection Secret

Generaing data using RAND command
    Generated data:
        E3 C0 60 F4 6F 48 70 83 F8 C8 7F 1C B0 25 93 F0
        2B 5B F4 D8 4E 50 E8 B1 3C 5E 94 1E 76 4A 11 BF

Encrypted Write Command:
    Writing data to slot 3
    Write Success

Encrypted Read Command:
    Reading data stored in slot 3
    Read data:
        E3 C0 60 F4 6F 48 70 83 F8 C8 7F 1C B0 25 93 F0
        2B 5B F4 D8 4E 50 E8 B1 3C 5E 94 1E 76 4A 11 BF

Verifing read data matches written data:
    Data Matches!

Done
```

ECDSA 签名验证示例

Sign 命令使用 ECDSA 算法和 Slot 中的私钥生成签名。

Verify 命令接受 ECDSA 签名，并验证该签名是否由输入消息给定的摘要和公钥正确生成的。

这个例子演示了 ECC Sign 和 Verify 命令在受支持的情况下的使用加密身份验证设备以及主机端创建和验证签名的步骤。

在这个例子中:

- 生成随机消息
- 消息使用私钥签名
- 签名消息使用关联的公钥验证

签名的创建和验证可以由设备或主机根据脚本参数执行。

```
$ python .\sign_verify.py

Sign/Verify Example

Signing Public key:
    2A 51 36 3C 62 10 6A D5 27 8E 0B 72 ED 3A A1 B9
    4E DF 3B C8 45 82 44 93 9E 18 C3 36 FB 7F A3 2C
    F5 B0 47 0E 3D CE 55 7D 99 A0 56 34 BE 43 59 36
    50 82 93 49 58 7B F4 2B 99 DA 3C 33 12 19 99 82

Message Digest:
    09 BA 4E F5 83 A1 E8 19 19 1F FD 4B 86 D5 55 0D
    27 6B 4A C0 D3 54 E1 C4 FD CC 47 D6 D8 8C DC 33

Signing the Message Digest
    Signing with device

Signature:
    22 3C 80 CE F4 2C DB 55 FB 8D 5B 57 03 FB 3A 4C
    0A EB 98 90 85 DC 33 7B D7 3B E8 4B B3 3E 36 C5
    E0 89 C6 F6 AA 80 DC 9F 37 13 97 2D 18 ED 3D BF
    62 10 0A CC 68 6B 1A 45 21 71 2E 5C 14 86 F9 98
```

```
Verifing the signature:
    Verifying with host
    Signature is valid!

Done
```