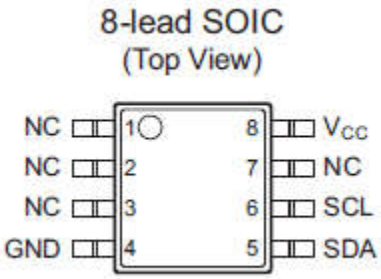
ATSHA204A 加密芯片攻略——使用篇

ATSHA204A 加密芯片是 ATMEL 公司研发的一款高安全性的，功能丰富的加密 IC，使用 SHA-256 算法进行加密操作 ,内置 16*32 字节的 slot(EEPROM)可以存储用户数据和秘钥，唯一的 9 字节序列号用于区分其他芯片，还有 512bits 的 OTP 区用于保存一些固定信息。芯片对外有 2 中通信方式，分别是单 bus 和 i2c 方式，本文所演示的全部使用 I2C 方式，但所有功能使用单 bus 方式也可以实现。

芯片管脚图非常简单，如下：



这个是 ATSHA204A 的 SOIC 封装，只需要电源，地，I2C 的 SCL 和 SDA 这 4 根线即可，SCL 和 SDA 需要上拉电阻。

首先我先讲一下 SHA-256 到底是个什么东东。

安全散列算法 SHA（Secure Hash Algorithm）是美国国家安全局 （NSA） 设计，美国国家标准与技术研究院（NIST） 发布的一系列密码散列函数，包括 SHA-1、SHA-224、SHA-256、SHA-384 和 SHA-512 等变体。主要适用于数字签名标准（DigitalSignature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA），

SHA 算法最主要的特点就是，任意长度的输入能生成固定长度的输出，并且从输出的结果中不能还原输入的内容，而且要找到不同两个输入导致相同输出的情况在计算上不能实现。

ATSHA204A 使用的 SHA256 算法，能输出 32 字节(256 位)的固定长度输出，基于 SHA

算法的特性，我们完全可以做到程序启动的秘钥验证，或者服务端和客户端的秘钥验证来保护我们产品。

废话说了一些，以下看一下如何使用 I2C 操作这款芯片，本文是假设已经对芯片进行默认配置的情况下来进行操作的，默认配置为如下：

秘钥保存在 slot0，SlotConfig0.Bit12 = 0(表示在运行 DeriveKey Command 时，使用目标 slot 区的秘钥进行 SHA256 计算而不是使用指定的 parentkey)。然后对芯片的锁定，锁定后不能再对配置区进行修改，而数据区可根据配置来使用其他命令更新，详细请见下篇——配置篇。

对 ATSHA204A 操作时，首先要对设备进行唤醒操作，唤醒操作如下：

SDA 线保持低电平 60us，芯片退出休眠状态，然后再等待 2.5ms 后，才能开始接受命令。

在发送 I2C 数据的时候，发送完设备地址后，应该发送一个 word_address 来指示这个包有何用处。如下

Table 6-2. Word Address Values

Name	Value	Description
Reset	0x00	Reset the address counter. The next read or write transaction will start with the beginning of the I/O buffer.
Sleep (Low Power)	0x01	The ATSHA204A goes into the low-power sleep mode and ignores all subsequent I/O transitions until the next Wake flag. The entire volatile state of the device is reset.
Idle	0x02	The ATSHA204A goes into the idle state and ignores all subsequent I/O transitions until the next Wake flag. The contents of TempKey and RNG Seed registers are retained.
Command	0x03	Write subsequent bytes to sequential addresses in the input command buffer that follow previous writes. This is the normal operation.
Reserved	0x04 – 0xFF	These addresses should not be sent to the device.

这里写图片描述

在芯片手册中已经注明了其他命令的作用，对芯片的操作主要使用 command，即 0x03 来操作。

command 有很多种，在验证秘钥的时候往往需要使用几种命令。

Table 8-4. Command Opcodes, Short Descriptions, and Execution Times

Command	Opcode	Description	Typ. Exec. Time(1), ms	Max. Exec. Time(2), ms
DeriveKey	0x1C	Derive a target key value from the target or parent key.	14	62
DevRev	0x30	Return device revision information.	0.4	2
GenDig	0x15	Generate a data protection digest from a random or input seed and a key.	11	43
HMAC	0x11	Calculate response from key and other internal data using HMAC/SHA-256.	27	69
CheckMac	0x28	Verify a MAC calculated on another Atmel CryptoAuthentication device.	12	38
Lock	0x17	Prevent further modifications to a zone of the device.	5	24
MAC	0x08	Calculate response from key and other internal data using SHA-256.	12	35
Nonce	0x16	Generate a 32-byte random number and an internally stored nonce.	22	60
Pause	0x01	Selectively put just one device on a shared bus into the idle state.	0.4	2
Random	0x1B	Generate a random number.	11	50
Read	0x02	Read four bytes from the device, with or without authentication and encryption.	0.4	4
SHA	0x47	Calculate a SHA256 digest for any system purpose.	11	22
UpdateExtra	0x20	Update bytes 84 or 85 within the Configuration zone after the Configuration zone is locked.	8	12
Write	0x12	Write 4 or 32 bytes to the device, with or without authentication and encryption.	4	42

一般来说，我们用到的有这几个

DerriveKey：更新一个 slot 区中的秘钥命令。

Nonce:使用随机数来更新 TempKey 中的值

CheckMAC：验证一个摘要是否计算正确，返回的是 Bool 值类型(0 是正确,1 是错误)

MAC：使用 SHA-256 计算并返回一个摘要。

Read:读取芯片区域中的数据。

Write:写入芯片区域中的数据。(这个命令一般用在配置时)

其他命令相对来说使用比较少，而且其他有些命令也相对简单好理解，但并不意味它们是没

有作用的，因篇幅问题，本文介绍几个命令。

Nonce 命令：

从字面命令上来理解，这里是随机命令的意思，的确如此。这个命令目的在于更新芯片中的 TempKey 值，什么是 TempKey，可以这样理解，TempKey 就类似于我们程序中的临时变量，来保存一些中间结果用的，但这个 TempKey 值不能被访问，只能芯片内部自己使用。

首先介绍 Nonce 命令也因为它是其他多数命令执行前都要执行的一个命令。

Table 8-25. Input Parameters

	Name	Size	Notes
Opcode	Nonce	1	0x16.
Param1	Mode	1	Controls the mechanism of the internal RNG and seed update.
Param2	Zero	2	Must be 0x0000.
Data	NumIn	20,32	Input value from system.

Nonce 命令如上

比较关注于 Param1 mode，其他都是固定的值

mode 介绍如下

Table 8-27. Mode Encoding

Bits	Meaning
2 – 7	Must be zero.
0 – 1	If zero, then combine the new random number with NumIn, store in TempKey. Automatically update EEPROM seed only if necessary prior to random number generation. Recommended for highest security. If one, then combine the new random number with NumIn, store in TempKey. Generate random number using existing EEPROM seed, do *not* update EEPROM seed. If two, then is it invalid. If three, then operate in the pass-through mode and write TempKey with NumIn.

好理解，0 或者 1 的时候，需要我们输入一个 20 字节的随机数，然后更新芯片内的 seed，更新 seed 能使内部产生的随机数质量更高。2 值不被允许，3 的话就要求输入 32 字节的随机数，直接写在 TempKey 中。

我们配置如下

opcode = 0x16,

mode = 0x01,

zero = 0x0000,

numin = 20 字节随机数

当命令正确执行的时候

将返回 32 字节的由芯片产生的 randout 随机数，如果 mode = 0x03，只返回 Bool 值。

MAC 命令：

使用 MAC 命令可以让 ATSHA204 生成一个摘要。

如图

Table 8-22. Input Parameters

	Name	Size	Notes
Opcode	MAC	1	0x08.
Param1	Mode	1	Controls which fields within the device are used in the message.
Param2	SlotID	2	Which internal key is to be used to generate the response. Bits 0:3 only are used to select a slot but all 16 bits are used in the SHA-256 message.
Data	Challenge	0 or 32	Input portion of message to be digested, ignored if Mode:0 is one.

比较关注的是 Param1 mode

Table 8-24. Mode Encoding

Bits	Meaning
7	Must be zero.
6	If set, include the 48 bits SN[2:3] and SN[4:7] in the message; otherwise, the corresponding message bits are set to zero.
5	Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored.
4	Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message; otherwise, the corresponding message bits are set to zero.
3	Must be zero.
2	If either Mode:0 or Mode:1 are set, Mode:2 must match the value in TempKey.SourceFlag or the command will return an error.
1	If zero, then the first 32 bytes of the SHA message are loaded from one of the data slots. If one, then the first 32 bytes are filled with TempKey.
0	If zero, then the second 32 bytes of the SHA message are taken from the input Challenge parameter. If one, then the second 32 bytes are filled with the value in TempKey. This mode is recommended for all use.

好像很复杂，不用担心，复杂的问题我们把他简化就好了。

mode.bit0 控制的是使用 Tempkey 计算还是直接使用 Data 中的 Challenge 来计算返回的 digest。

mode.bit1 控制的是使用 slot 区中的值计算还是 Tempkey 中的值来计算返回的 digest。

mode.bit2 如果 bit0 或 bit2 设置了，那么此位要匹配 Tempkey.sourceflag 位匹配。也就是说，使用了 nonce 命令的话，如果 nonce 命令中的 nonce.mode == 0x00 或者 0x01，这里就等于 0, 如果 nonce.mode == 0x03，这里就等于 1。

mode.bit3 = 0

mode.bit4 和 mode.bit5 是选择是否使用 OTP 中的的值来参与计算，个人觉得意义不大而且增加了配置时的复杂程度。设置为 0 就好了。

mode.bit6 是使用序列号来参与计算，可以设置。

mode.bit7 = 0

本人设置如下

opcode = 0x08

mode = 0x01|0x06

slotid = 0

challenge = 0

发送过去后，如果执行正常，会返回一个 32 字节的 digest 结果，这个结果的计算如下

The message that will be hashed with the SHA-256 algorithm consists of the following information:

32 bytes	key[SlotID] or TempKey (See Table 8-24.)
32 bytes	Challenge or TempKey (See Table 8-24.)
1 byte	Opcode (Always 0x08.)
1 byte	Mode
2 bytes	Param2
8 bytes	OTP[0:7] or zeros (See Table 8-24.)
3 bytes	OTP[8:10] or zeros (See Table 8-24.)
1 byte	SN[8]
4 bytes	SN[4:7] or zeros (See Table 8-24.)
2 bytes	SN[0:1]
2 bytes	SN[2:3] or zeros (See Table 8-24.)

芯片内部会根据一系列的值去计算这个 32 字节的 digest。

有什么用？

看到这里大家可能有这个疑问，返回这个值有什么用？我要拿来做什么？

别急，我为大家讲一个防程序拷贝的流程措施。

假设我们很辛苦的开发了程序，但是又不想被别人盗用劳动成果（执行文件被拷出来，抄个板继续使用）。我们就要在启动的时候验证一下是否通关，不通关就不往下走。流程如下：

首先我们 ATSHA204A 芯片的 slot0 中写入一个 32 字节的秘钥，设置为不可读。这个秘钥只有几个人知道，并且不外泄，主控 MCU 与芯片相连，在启动时候，首先我们要获取芯片序列号，为后面计算 MAC。然后我们发送一个 nonce 命令给 ATSHA204A，让其更新内部的 TempKey 中的值，MCU 内部也根据 nonce 的模式计算一下 Tempkey 的值，我们称之为 host_tempkey，如果一切正常，那么 Tempkey 和 host_tempkey 中的值是一致的，接下来 MCU 自己就可以根据 host_tempkey、序列号、以及秘钥计算，出一个摘要 digest，我们称之为 digest1，接下来给 ATSHA204A 发送 MAC 命令，ATSHA204A 根据存储在 slot 区域中的秘钥来计算 digest2 并返回给 MCU，MCU 判断 digest1 和 digest2 是否匹配，不匹配将程序挂起即可。

有几个关键点要注意一下：

1、在 I2C 总线上并没有传送秘钥，而是传送根据秘钥和一些列的值计算出来的 digest，因

此在总线上截获秘钥不可实现。

2、在自己的 MCU 上也存了秘钥，别人获取了 16 进制文件后是否有可能得到秘钥，我只能说，的确存了，但是对于一个庞大的程序 hex 文件中寻找一个 32 字节的秘钥，岂不是大海捞针？

3、在 MCU 上自己也要使用 SHA-256 算法计算摘要，怎么实现，这个问题放心，在 ATMEL 官网上有已经实现好的库，可以直接使用，但是这个库在移植上有些问题需要改一下，本人已经移植好了基于 stm32f103 的版本，完全正常使用，并且带有中文注释，移植只需要修改几个接口即可，有想业务合作的可以私信。

4、在使用 nonce 时，要发送一个随机数给 ATSHA204A，如果担心随机数质量不好，可以使用 ATSHA204A 的 Random 命令，就可以返回一个 32 字节的高质量的随机数来使用，也可以用作其他用途。

5、是否有破解的可能？对于一些小单片机（51 之类的），的确好破，反汇编后，跳过验证步骤即可，但是 stm32 之类的强大功能单片机就不是那么回事，首先如果你能反汇编，也得找出验证步骤在哪里，其次，我也不知道 stm32 有没人做过反汇编，因此，只要秘钥不泄露，要破解起来个人觉得也很难。毕竟这个可是军方用过的东西。

HMAC 命令：

这个命令与 MAC 命令几乎一样。只不过使用的算法不一致，而且根据芯片手册上介绍，效率也低。因此这里不在论述。

CheckMAC 命令：

这个命令也是带有验证功能的，简单来讲，就是让你计算好 digest，发给 ATSHA204A，然后 ATSHA204A 告诉你这个 digest 算的对不对，返回 bool 值

如下

Table 8-8. Input parameters

	Name	Size	Notes
Opcode	CHECKMAC	1	0x28
Param1	Mode	1	Bit 0: Source of the second 32-bytes of the SHA message. 0: ClientChal parameter 1: TempKey Bit 1: Source of the first 32-bytes of the SHA message. 0: Slot[SlotID] 1: TempKey Bit 2: If TempKey is used, this bit must match the value of TempKey.SourceFlag. Bit 3-4: Must be zero. Bit 5: 8-bytes of SHA message. 0: zeros 1: OTP zone Bits 6-7: Must be zero.
Param2	SlotID	2	Which internal slot is to be used to generate the response. Only bits 0:3 are used.
Data1	ClientChal	32	Challenge sent to Client. (*Must* appear in the input stream).
Data2	ClientResp	32	Response generated by the Client.
Data3	OtherData	13	Remaining constant data needed for response calculation.

看了之前命令介绍，这里大家都应该很明白了，

为了方便起见，例子如下

mode = 0

SlotID = 0

ClientChal = 32 字节，这里 MCU 可以发送一个 32 字节的随机数

ClientResp = 32 字节，这里就是 MCU 自己计算好的 digest，发给 ATSHA204

OtherData = 13 字节如下

Table 8-10. OtherData

Size	CheckMac	MAC	Notes
1	OtherData[0]	OpCode	MAC OpCode = 08
1	OtherData[1]	Mode	Mode used for MAC command.
2	OtherData[2:3]	SlotID	SlotID used for MAC command.
3	OtherData[4:6]	OTP[8:10]	OTP[8:10] used for MAC command. (Useful for Legacy.)
4	OtherData[7:10]	SN[4:7]	SN[4:7] used for MAC command. (Unique per Client.)
2	OtherData[11:12]	SN[2:3]	SN[2:3] used for MAC command. (Unique per Client.)

不使用 OTP，有关 OTP 的参数全部给 0 就好了

ClientResp 的计算要依据下面

The message that will be hashed with the SHA-256 algorithm consists of the following information:

32 bytes	key[SlotID] or TempKey (Depending on the mode.)
32 bytes	ClientChal or TempKey (Depending on the mode.)
4 bytes	OtherData[0:3]
8 bytes	OTP[0:7] or zeros (Depending on the mode.)
3 bytes	OtherData[4:6]
1 byte	SN[8]
4 bytes	OtherData[7:10]
2 bytes	SN[0:1]
2 bytes	OtherData[11:12]

在官方库里面也封装好了相关操作，直接拿来用即可。

如果一切正常，ATSHA204A 返回的 0，计算 digest 不对就返回 1。

这个 CheckMAC 命令的作用，我个人觉得其中一个作业就是是去验证密码。比如有一个主设备，它把秘钥存在 slot0 里面，有一些从设备，要输入正确的密码才能使用主设备提供的服务，就可以用这种方式，这种方式在总线上也不会传输密码，而且把密码存在 ATSHA204A 里面更安全（其他诸如 flash,eeprom 容易被盗取）。

DeriveKey 命令

这个命令从字面上理解，就是变化 Key 的命令意思，的确，可以通过这个命令去更新 slotid 中的 key。

这个命令内容简单，但是理解起来比较费劲，根据手册，介绍如下

8.5.6 DeriveKey Command

The device combines the current value of a key with the Nonce stored in TempKey using SHA-256, and places the result into the target key slot. SlotConfig[TargetKey].Bit 13 must be set or DeriveKey will return an error.

If SlotConfig[TargetKey].Bit12 is zero, the source key that will be combined with TempKey is the target key specified in the command line (Roll-Key operation). If SlotConfig[TargetKey].Bit12 is one, the source key is the parent key of the target key, which is found in SlotConfig[TargetKey].WriteKey (Create Key operation).

Prior to execution of the DeriveKey command, the Nonce command must have been run to create a valid nonce in TempKey. Depending upon the state of bit two of the input mode, this nonce would have been created with the internal RNG, or it would have been fixed.

If SlotConfig[TargetKey].Bit15 is set, an input MAC must be present and had been computed as follows:

SHA-256(ParentKey, Opcode, Param1, Param2, SN[8], SN[0:1])
where the ParentKey ID is always SlotConfig[TargetKey].WriteKey.

If SlotConfig[TargetKey].Bit12 or SlotConfig[TargetKey].Bit15 is set and SlotConfig[ParentKey].SingleUse is also set, DeriveKey returns an error if UseFlag[ParentKey] is 0x00. DeriveKey ignores SingleUse and UseFlag for the target key if SlotConfig[TargetKey].Bit12 and SlotConfig[TargetKey].Bit15 are both zero.

For slots 0 thru 7 only, if input parsing and the optional MAC check succeed, UseFlag[TargetKey] gets set to 0xFF and UpdateCount[TargetKey] is incremented. If UpdateCount currently has a value of 255, then it wraps to zero. If the command fails for any reason, these bytes will not be updated. The value of UpdateCount may be corrupted if power is interrupted during the execution of DeriveKey.

Note: If the source and target keys are the same, there is a risk of permanent loss of the key value if power is interrupted during the Write operation. If the configuration bits permit it, then the key slot may be recovered using an authenticated and encrypted write based upon the parent key.

简单讲 ,就是 slot.slotconfig[TargetKey].bit13 要设置成 1 ,否则 DeriveKey 将返回错误，

然后如果 slot.slotconfig[TargetKey].bit12 为 0

要更新的秘钥值将是 TempKey 和原先的秘钥值计算出来的，

slot.slotconfig[TargetKey].bit12 为 1，则是 TempKey 和

slot.slotconfig[TargetKey].WriteKey 中指定的 SlotID 的值计算出来的。

另外，如果 SlotConfig[TargetKey].bit15 设置了的话，还需要输入

一个 MAC 摘要，即如下

SHA-256(ParentKey, Opcode, Param1, Param2, SN[8], SN[0:1])
where the ParentKey ID is always SlotConfig[TargetKey].WriteKey.

以上都是配置阶段配置好的，不能修改，而发送这个命令时带如下参数即可，

Table 8-11. Input Parameters

	Name	Size	Notes
Opcode	DERIVEKEY	1	0x1C
Param1	Random	1	Bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error. Bits 0:1, 3:7: Must be zero.
Param2	TargetKey	2	Key slot to be written.
Data	Mac	0 or 32	Optional MAC used to validate operation.

Table 8-12. Output parameter

Name	Size	Notes
Success	1	Upon successful completion, the ATSHA204A returns a value of zero.

因为要用到 TempKey，所以 Nonce 在 DeriveKey 命令执行前，需要被正确的执行，才能更新到 TempKey 中的值。

新的秘钥由以下计算出来。

The key written to the target slot is the result of a SHA-256 of the following message:

32 bytes	Target or parent key (depending on SlotConfig Bit12)
1 byte	Opcode
1 byte	Param1
2 bytes	Param2
1 byte	SN[8]
2 bytes	SN[0:1]
25 bytes	Zeros
32 bytes	TempKey.value

而 MCU 自己也可以计算出这个新的秘钥作为使用。

当我们原有的秘钥泄露的时候，可以使用这个命令来产生新的秘钥继续使用。

另外如果 slot.slotconfig[TargetKey].bit12 为 1 和 SlotConfig[TargetKey].Bit15 为 1 还有 SlotConfig[ParentKey].SingleUse 为 1 的时候(ParentKey 为 slot.slotconfig[TargetKey].WriteKey)，则 UseFlag[ParentKey]为 0 的时候，DeriveKey 将返回失败，也就是说，我个人理解 UseFlag[ParentKey]就是指示剩余使用的次数，一旦这个次数为 0，那么将不能使用这个 ParentKey 去生成其他 TargetKey。

Read 和 Write 命令：

这两个命令在配置阶段用的较多，这两个命令在我的下一篇，配置篇将会讲到。

配置篇地址为：http://blog.csdn.net/a5882230/article/details/52214845

至此，ATSHA204A 芯片的基本使用也介绍完毕，基于这么多种命令，我们可以把这个芯片用在很多地方，如防抄板，密码验证，消耗产品的匹对等等，当然还有其他一些功能这篇文章没有介绍到，感兴趣的朋友可以去完整的读一遍 datesheet，不过现在只有英文版的。

附录

鉴于官方的烧录器价格非常感人 ,而某宝上也没有便宜的烧录器 ,本人亲自做了两种烧录器，分别是在线烧录并可调试，另外一种就是脱机带屏显示烧录器，可以进行小批量的烧录，下面是烧录器的样式：

有需要的朋友可以到以下链接看看

https://item.taobao.com/item.htm?spm=a230r.1.14.22.oblrCT&id=550477591352&ns=1&abbucket=10#detail

另外一款脱机烧录带 oled 屏显示，支持一键下载，一键烧写，一键锁定，烧录高效，有兴趣的朋友可以到以下链接看看

https://item.taobao.com/item.htm?spm=a230r.1.14.44.oblrCT&id=553579727124&ns=1&abbucket=10#detail