应用笔记 157
SHA iButton API 概述

绪论

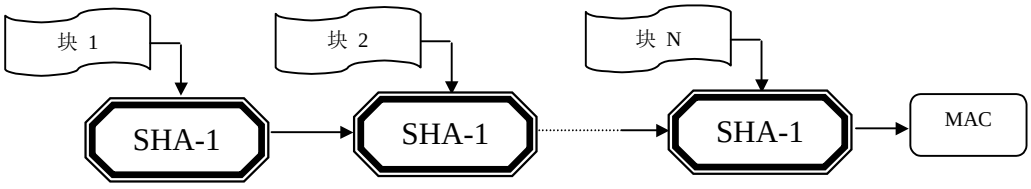
Dallas SHA iButton® (DS1963S) 是一个智能令牌，具有很高的安全性，并支持多种服务。本文简要介绍了使用 SHA iButton 实现数字认证和交易的应用，并以一个示范服务程序为例来具体说明安装服务数据和处理电子交易的步骤。本文还论述了主要 API 的调用方法，介绍了命令注释列表和实现各种 API 的数据流程。

SHA iButton 适用于多种应用：

- 批量运输
- 电子门锁
- 门镜控制
- 计算机和网络访问控制
- 电话付费
- 停车计费器
- 预付费表
- 自动售货机
- 软件授权

什么是 SHA?

SHA（安全散列算法）经过加密专家多年来的发展和改进，已经成为公认的最安全、并被广泛使用的散列算法之一。在 SHA iButton（Dallas Semiconductor 的型号为 DS1963S）中实现的 SHA 算法是 SHA-1，它符合联邦信息发行标准 180-1（即 FIPS 180-1）。杂散函数可以简单的理解为：取一串输入码（称为预映射或信息）并把它转化为固定的、较短长度的输出序列（称为散列值、信息摘要或信息认证代码）的过程。由于 SHA 是对压缩后的信息摘要进行检查，它对检测输入序列中的错误或变化是非常有效的。单向散列函数的特征是产生摘要容易，但从给定的摘要反求出输入信息非常困难。单向散列函数的安全性就在于其产生摘要的操作过程具有较强的单向性。在输入序列中嵌入密码，任何人在不知道密码的情况下都不能产生正确的摘要，从而保证其安全性。SHA 将输入流按照每块 512 位（64 个字节）分块，并产生 20 个字节、被称为信息认证代码（MAC）或信息摘要的输出。如果输入长度不是 512 位，算法就把它凑成最接近于 512 的整倍数值。下图说明了由多个输入块（512 位）产生最后的 MAC 的链状图。

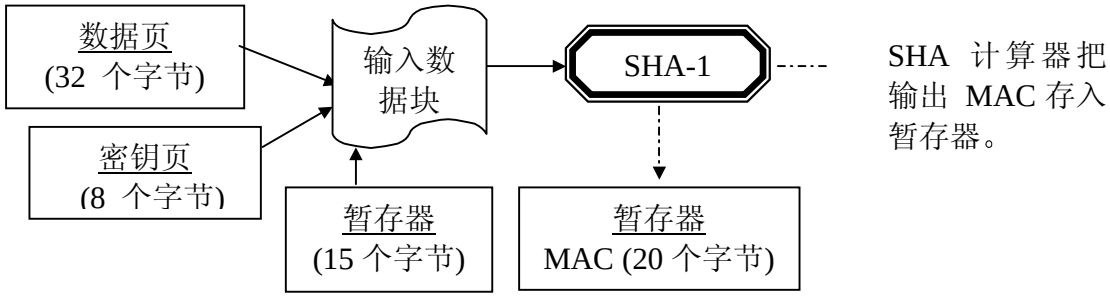


iButton 是 Dallas Semiconductor 的注册商标。

Dallas SHA iButton（DS1963S）中的 SHA

Dallas SHA iButton（DS1963S）在专门的快速电路中执行 SHA-1 计算，它可以在 1ms 之内完成一次 SHA-1 计算。SHA-1 引擎的输入被限制为一块。通过设计，SHA-1 引擎的输入数据块包括存储在读/写数据页（32 个字节）、暂存器中的用户数据（15 个字节）和服务供应商预装在不可读取的加密页中的密钥（8 个字节）（参见图 1）。计算结果（MAC）保存在暂存器中用于后续操作。由于对特定 SHA 操作的依赖性，该输出 MAC 在通过外部访问的暂存器中不一定要隐藏起来。

SHA iButton 中 MAC 生成器 图 1



SHA iButton 的特性

SHA iButton 是一个只需最少硬件就能访问大约 4k 位读/写数据的载体。通过激活集成的 512 位 SHA-1 引擎算出基于器件内存储信息的 160 位的信息认证代码（MAC）。数据依照 1-Wire®协议串行传输，该协议只需一根数据连线和一根地线。每个 SHA iButton 利用其相应的密钥和页计数器能够同时服务于 8 个独立的应用中。SHA iButton 也可以作为协处理器保护系统密钥的安全，也可以协助本地交易主机计算用户令牌认证和确认应用数据所需要的 MAC。SHA iButton 和其它 iButton 一样有附加的存储区域，称为暂存器，向存储器写数据时用作缓冲器。SHA iButton 的暂存器还用于向 SHA-1 引擎注入数据段或接收/比较信息认证代码。向 iButton 写数据的时候，数据首先被写到暂存器中，在这里，数据能被读回并验证是否有通信错误。数据被验证之后，复制暂存器命令把数据传到目标存储器位置。这一过程在没有可靠的电接触环境下可以确保数据的完整性。

DS1963S SHA iButton 有如下特殊性能：

- 4096 位可读/写的非易失存储器，分成 16 页、每页 256 位
- 16 个存储页中的 8 个有独自的 64 位密钥和 32 位只读、可回溯写次数计数器
- 内置 512 位 SHA-1 引擎
- 作为用户令牌，器件能同时支持 8 个独立的服务
- 可用作协处理器存储系统密钥和计算用户令牌认证、确认应用数据所需要的 MAC

1-Wire 是 Dallas Semiconductor 的注册商标。

SHA iButton 存储器图

通用的读/写访问数据存储器（16 页，每页 32 个字节）

页码	地址范围 (TA1 TA2)	密钥号	计数器值	计数器递增
0	0000H 至 001FH	0	0	无
1	0020H 至 003FH	1	1	无
2	0040H 至 005FH	2	2	无
3	0060H 至 007FH	3	3	无
4	0080H 至 009FH	4	4	无
5	00A0H 至 00BFH	5	5	无
6	00C0H 至 00DFH	6	6	无
7	00E0H 至 00FFH	7	7	无
8	0100H 至 011FH	0	0	写数据时
9	0120H 至 013FH	1	1	写数据时
10	0140H 至 015FH	2	2	写数据时
11	0160H 至 017FH	3	3	写数据时
12	0180H 至 019FH	4	4	写数据时
13	01A0H 至 01BFH	5	5	写数据时
14	01C0H 至 01DFH	6	6	写数据时
15	01E0H 至 01FFH	7	7	写数据时

不可读取的加密存储器（八个 64 位密钥）

页码	地址范围 (TA1 TA2)	说明
16	0200H 至 0207H	密钥 0
	0208H 至 020FH	密钥 1
	0210H 至 0217H	密钥 2
	0218H 至 021FH	密钥 3
17	0220H 至 0227H	密钥 4
	0228H 至 022FH	密钥 5
	0230H 至 0237H	密钥 6
	0238H 至 023FH	密钥 7

值得注意的是只有向第 8 到第 15 页写数据时，页计数器才会递增。在特定操作中一对匹配数据页共用一个计数器和密钥。例如，第 0 页和第 8 页共用 0 号计数器和 0 号密钥。向第 8 页写数据时 0 号计数器增 1，而向第 0 页写数据时对 0 号计数器没有影响。在第 0 页或第 8 页执行 Read Authenticated Page（读鉴别页）命令时都将返回 0 号计数器的值，计算 MAC 都将使用密钥 0。

只读计数存储器（九个 32 位计数器）

页码	地址范围 (TA1 TA2)	说明
19	0260H 至 0263H	第 8 页的写次数计数器
	0264H 至 0267H	第 9 页的写次数计数器
	0268H 至 026BH	第 10 页的写次数计数器
	026CH 至 026FH	第 11 页的写次数计数器
	0270H 至 0273H	第 12 页的写次数计数器
	0274H 至 0277H	第 13 页的写次数计数器
	0278H 至 027BH	第 14 页的写次数计数器
	027CH 至 027FH	第 15 页的写次数计数器
20	0280H 至 0283H	密钥 0 的写次数计数器
	0284H 至 0287H	密钥 1 的写次数计数器
	0288H 至 028BH	密钥 2 的写次数计数器
	028CH 至 028FH	密钥 3 的写次数计数器
	0290H 至 0293H	密钥 4 的写次数计数器
	0294H 至 0297H	密钥 5 的写次数计数器
	0298H 至 029BH	密钥 6 的写次数计数器
	029CH 至 029FH	密钥 7 的写次数计数器
21	02A0H 至 02A3H	PRNG 计数器 (SHA 引擎实现计数)

在电子付费应用中使用 SHA iButton

SHA iButton 既可以用作用户令牌，又可以用作协处理器。协处理器可以完成交易会话期间的器件认证和验证应用数据所需要的 MAC 计算，无需执行 SHA 计算代码、有效缩短了开发周期。把 SHA iButton 用作协处理器的另一个好处是可以把系统密钥存储在钮扣内，而钮扣不能够被检测，也就不会泄漏密钥。在本文中我们假设 SHA iButton 协处理器用于本地主机，执行必要的器件认证和服务数据验证。本文将用到的一些术语定义如下：

本地主机－包含必要组件的硬件单元，它能利用用户 SHA iButton（或其它 ePurse 令牌）完成电子交易。从功能上讲，本地主机包含三个关键部件：交易控制单元（典型的微处理器）；诸如显示器和令牌接收器的用户接口；协处理器。

地址编码（AN）、ROM ID、注册号、序列号－可替代使用，代表 DS1963S 或其它任何 1-Wire 器件内部由工厂激光刻入的、全球唯一的 64 位码。

交易控制单元（TCU）－在协处理器和用户令牌之间进行通信的组件（典型的微处理器），实现器件认证和服务数据验证。

系统认证密钥、主认证密钥－可替代使用，安装在协处理器中的密钥、用于鉴别用户器件。

器件认证密钥－安装在用户 iButton 中的密钥，本地主机可以验证其是否属于该系统。通过绑定系统认证密钥与用户器件的地址编码保证了每个用户器件认证密钥的唯一性。

系统签名密钥，主签名密钥－可替代使用，代表安装在协处理器中用于签名和验证服务数据的密钥。

签名－由服务数据、系统签名密钥和其它服务或器件指定数据计算出的信息认证代码（MAC）。签名通常嵌入于服务数据页中，使本地主机能快速地验证服务数据。

器件数据、用户数据、应用数据、账户数据、交易数据－代表诸如访问控制或电子付费应用的服务数据。

用户器件、用户令牌－数字认证或电子付费应用中服务数据的数字载体。本文凡是提到用户器件或用户令牌时都指 DS1963S SHA iButton。

协处理器－本文协处理器是一个计算单元，能够完成交易过程中所必要的 MAC 计算。

服务安装

为了使用器件认证和数据验证的 SHA iButton，必须在协处理器和用户 iButton 中安装适当的数据和密钥——这个过程通常被称为服务初始化。协处理器内装有两个密钥：系统认证密钥和系统签名密钥。系统认证密钥用来对每个用户器件建立唯一的认证密钥和验证用户器件是否属于该系统。系统签名密钥用于产生签名和验证服务数据的有效性。用户 iButton 内只需安装一个密钥：唯一的器件认证密钥。值得重视的是用户器件（SHA iButton）的认证密钥是唯一的，它与系统认证密钥不同，利用用户的器件地址作为计算器件密钥输入的组成部分。

服务数据有两类：静态的和动态的。静态服务数据在交易期间从不发生改变，而动态服务数据在每次交易的过程中总是进行更新。比如，用于办公室或饭店的访问控制的服务数据含有定时的访问特权信息，这些数据需要进行验证，但同步处理器（本地主机）不对它进行修改。另一方面，自动售货机或停车计费器则需要从账户借记适当的金额，并对减少后的余额数进行再次数字签名，使其能够被其它本地主机加以验证。如果系统只是要知道一个用户器件是否属于该系统（经授权的用户），而不必根据服务数据的内容确定是否交易，就不必保护服务数据。

总的说来，电子付费系统包含以下步骤：
对于服务安装：

（1） 把系统认证密钥安装到 SHA iButton 协处理器

（2） 把系统签名密钥安装到 SHA iButton 协处理器（如果需要数据保护）

对于每个用户器件：

（3） 把用户器件认证密钥安装到用户 SHA iButton

（4） 把有签名的服务数据安装到用户 SHA iButton（如果需要数据保护）

对于处理交易：

（5） 使用 SHA iButton 协处理器完成用户器件认证

（6） 使用 SHA iButton 协处理器完成用户数据验证（如果需要数据保护）

器件认证

为了认证用户 SHA iButton，TCU 会请求协处理器计算一个随机质询[1]，并将其送回用户 iButton、请求它利用 Read Authenticated Page 命令计算应答 MAC。用户 iButton 利用质询、服务数据、自身的地址编码及其认证密钥计算应答 MAC。然后本地主机读取该应答 MAC、并与它随后的计算进行比较。为了验证应答 MAC，TCU 会请求协处理器利用用户器件地址编码和系统认证密钥首先重新计算用户 iButton 唯一的器件认证密钥。随后，TCU 请求协处理器利用重新计算的器件认证密钥和发送给用户 iButton 的质询码计算 MAC。协处理器计算出的 MAC 再和从用户 iButton 读取的应答 MAC 进行比较，以确定用户器件是否合法。这两步处理过程是必要的，因为本地主机不能读取用户器件的密钥，而且用户器件认证密钥与系统认证密钥不同。值得注意的是器件认证只需要用户器件携带有正确的认证密钥，并不关心服务数据的内容。

服务数据保护

服务数据保护可以通过嵌入系统产生的服务数据 MAC（签名）实现。利用系统签名密钥、服务数据、服务数据页码、页计数值和用户器件地址编码产生签名。利用签名可以检测出任何未经授权的变动，防止重放服务数据。为了验证服务数据，TCU 会令协处理器用系统签名密钥和所有运行期间获得的服务、用户器件数据重新计算 MAC（签名）。然后，MAC 将与嵌入的签名进行比较以确定数据的有效性。对于动态数据服务而言，服务数据更新后将计算一个新的签名来反映数据变更和新的页计数值。这个新的签名将嵌入到服务数据、并保存到用户器件中。

[1]该质询从某中意义上讲是动态的，是因为它受 SHA 引擎计数值的影响，每次运行 SHA 引擎计数器都会递增，而且用户 iButton 不可能收到同一质询两次。关于产生质询和密钥更详细的资料参见 AN152。

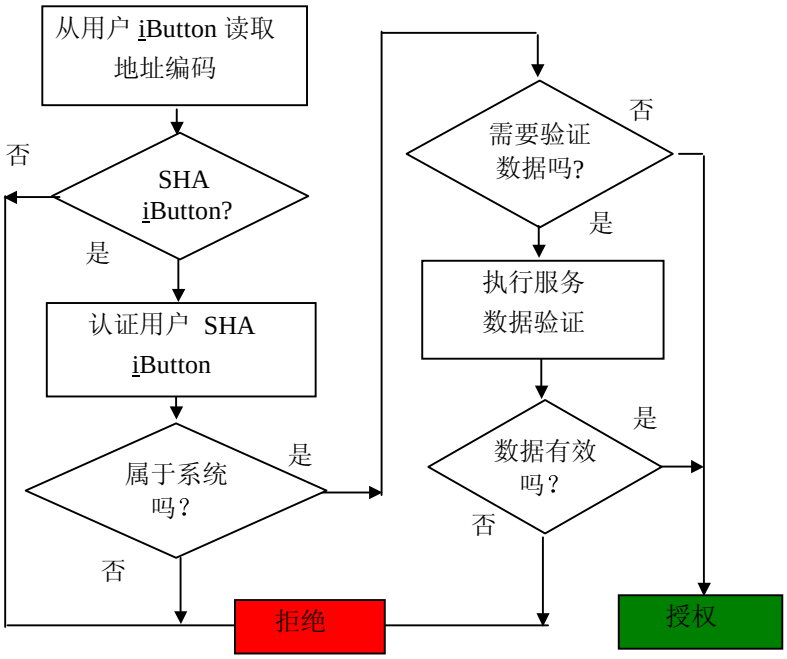
静态数据服务

静态数据服务只需进行器件认证和服务数据验证，以便做出服务的决定。器件认证代表验证用户器件是否属于系统的过程。器件认证在特定的访问控制应用中也许是唯一需要的手续。实现器件认证的方法多种多样：（a）读取器件的地址编码（AN），搜索数据库查看其是否属于系统；（b）执行一次质询和应答过程检测器件是否带有有效密钥；（c）其它方法。SHA iButton 用第二种方法认证器件。在这种质询与应答方法中，本地主机请求用户器件根据其隐藏的认证密钥、与密钥相链接的存储数据和本地主机提供的质询码计算一个应答 MAC。用户器件从不向外泄漏它的密钥。这种认证机制使得系统非常安全，尤其是在本地主机与用户器件之间的通信链路不安全的情况下更有益。此外，对于用户器件的每次服务请求本地主机都发出不同的质询，使每次应答 MAC 均不相同，这就使得截取的通信数据位毫无用途。

完成静态数据服务的典型步骤如下（参见图 2）

（1） 从用户 iButton 读取地址编码（AN）
（2） 如果不是 SHA iButton 则转到其它决定过程
（3） 通过质询和应答过程认证用户 iButton；如果用户 iButton 不属于系统则退出
（4） 需要验证服务数据吗？如果不需要，则提供服务。
（5） 核对服务数据是否有效；无效则退出
（6） 提供服务

静态数据服务判定树　　图 2

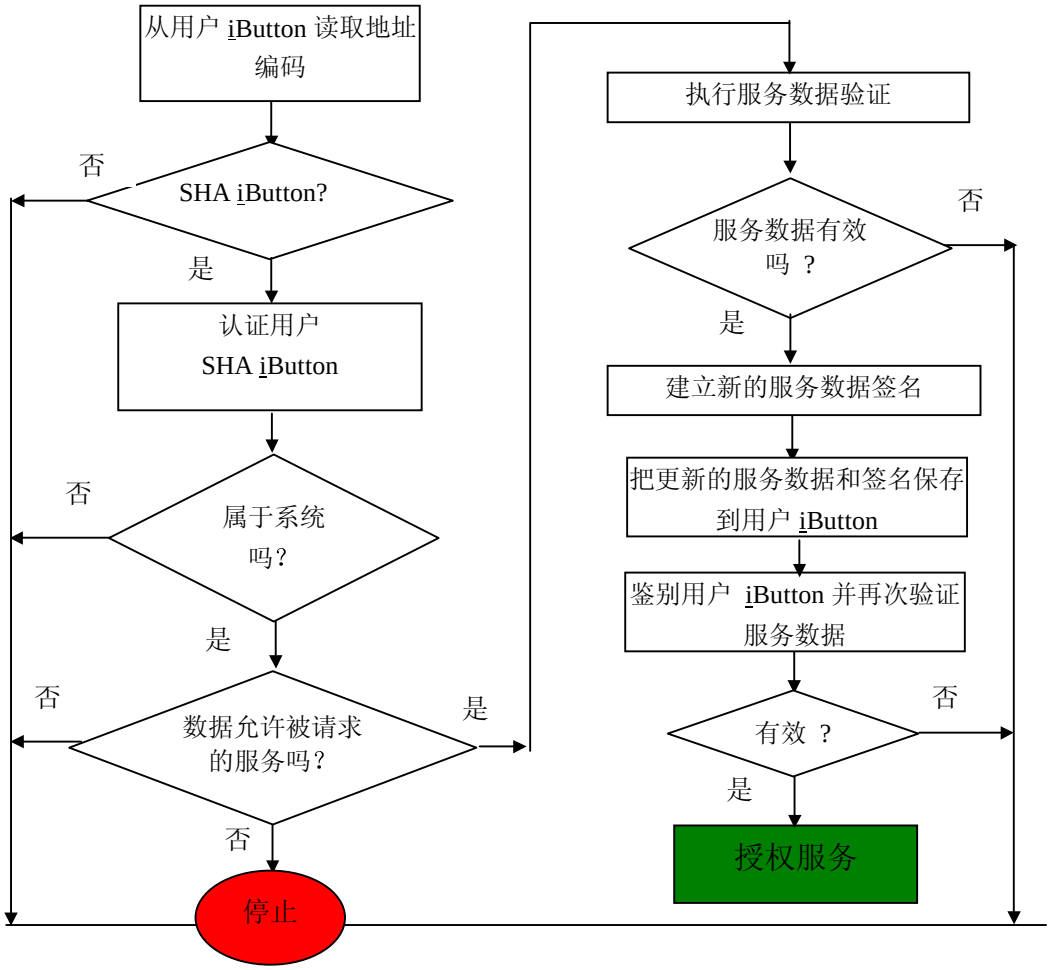


动态数据服务

除了在执行静态数据服务时的认证和数据验证外，利用动态数据的服务还需要额外的步骤来更改服务数据，重新建立签名和把重新签名的数据保存到用户 iButton。典型步骤概括如下：

（1） 从用户 iButton 读取地址编码（AN）
（2） 如果不是 SHA iButton，则转到其它决定过程
（3） 通过质询和应答过程认证用户 SHA iButton；如果用户 SHA iButton 不属于系统则退出
（4） 服务数据是否允许提供被请求的服务；不允许则退出
（5） 执行服务数据验证；如果无效则退出
（6） 更改服务数据并建立新的签名
（7） 把有签名的新服务数据存储到用户 iButton
（8） 再次执行用户器件认证，以确保用户器件中的原始服务数据被更新
（9） 提供服务

动态数据服务判定树　图 3



寄主多重服务

可以利用 1-Wire File Structure（OWFS[2]）来促成一个用户 iButton 中共存多重服务 — 每种服务数据存储在自身的服务文件中。为便于存取，多个文件按照目录表排列。本地主机在目录列表中为一次服务读取文件入口地址获取实际页的位置和用于服务的页数。实际服务数据取自服务文件。

特殊的文件可以存储在协处理器和其它 iButton 中，把它们当作特殊器件进行验证。例如，我们在 SHA iButton 中建立了一个名为 COPR.0 的文件、并把它当作协处理器进行验证，而且把完成一次交易的 TCU 所必须的全部数据存于该文件中。我们可以在 iButton 中建立另一个名为 COLL.102 的文件，把它作为移动数据收集器进行验证，移动数据收集器有权从本地主机获得数据，随后转储到中央数据库中。当交易控制单元（TCU）开始工作的时候，它从协处理器 iButton 读取信息准备处理交易。当 TCU 检测到有数据集合器与其连接时将执行必要的验证，输出被请求的数据。

协处理器文件

向 DS1963S SHA iButton 中安装适当的系统密钥和数据可将其配制成协处理器。对于没有数据验证的静态数据服务，协处理器中只安装了系统认证密钥。而动态数据服务或者需要数据验证的静态数据服务，则必须在协处理器中安装系统认证密钥和系统签名密钥。协处理器也能执行认证用户 iButton 和验证服务数据所必须的全部 MAC 计算。在我们的示范中，在 SHA iButton 中建立了一个充当协处理器的协处理器文件（COPR.0）。COPR.0 包含了交易控制单元为处理交易所必须的全部数据。

协处理器文件数据结构[3] 表 1

协处理器文件 COPR.0 的结构			
	字节数	抽样资料	备注
服务数据文件名	5	DLSM.102	只保存基本名(DLSM)和扩展名，而不保存分隔符（.）。扩展字节按照十六进制数保存（66H=102 d）。
签名页码	1	8	合法的选择是第 0 页或第 8 页。第 0 页用做文件目录表。
认证页码	1	7	从足够高的页码开始、为存储文件留出足够的页。
工作空间页码	1	9	在交易过程中，是协处理器建立用户器件唯一的认证密钥所必需的。

[2] 1-Wire File Structure（OWFS）为驻留在包括 iButton 在内的 1-Wire 器件中的数据提供目录结构。当它们在其它文件系统允许随机访问命名后的文件。按照 OWFS 的定义和规则、利用容量达 16M 字节的器件足以存储嵌套目录中的多重文件。这些器件被组织为 2…65535 页，每页 32…256 个字节。关于 OWFS 结构和配套支持的详细信息参见 Dallas Semiconductor/Maxim 应用笔记 114。

[3]在执行服务的示范中所有的数据字节按照最低有效位在前写入。

协处理器文件 COPR.0 的结构			
版本号	1	1	服务供应商利用版本号跟踪服务配置的变化。
安装数据代码	4	04 0E 00 63H	安装代码按照 M D YY 格式存储。YY = 1900 年以后的年份。比如 4/14/1999 = 04 0E 00 63。
器件认证密钥绑定数据	39	39FFH 字节	服务供应商用它将认证密钥绑定到用户 SHA iButton。该数据块的前 32 个字节写到数据页，剩下的 7 个字节在计算器件认证密钥时写到暂存器中。暂存器实际上包括 15 个输入到 SHA 引擎的字节。其余 8 个字节是页码（1 个字节）和不带有 CRC 校验字节的器件地址编码。
服务签名代码	3	3 00H 字节	该数据是写到暂存器中用来建立服务数据签名的 15 个字节的一部分。其余的 12 个字节是页码（1 个字节）、页计数值（4 个字节）和不带有 CRC 校验码的器件地址编码（7 个字节）。
Provider name length (svcNameLen)	1	20	服务供应商名字的长度，可调节。
Signature length (signLen)	1	20	用户账务页中签名块的长度。这个值可以缩短以便为其它数据（辅助数据）留出空间。
Auxiliary data length (auxDataLen)	1	0	辅助数据块的长度，可调。长度为零意味着没有附加数据。
服务提供者的名字	20	Dallas Semiconductor	服务供应商的名字或说明。它的长度受 *Provider name length* 限制。
签名初始值	20	20 00H 字节	初始数据块的长度必须与上述 *Signature length* 匹配。这些字节被置位替代用户数据页的签名块，以建立数据签名。
辅助数据	0		辅助数据的长度必须与上述 *Auxiliary data length* 匹配。
加密算法代码	1	0	用来说明为保护文件内容所采用的加密或者编码机制类型的标志。
DS1961S 标志	1	0	一个用于说明为协处理器安装密钥是否作了适当截取、以便配合 DS1961S 使用的布尔标记。
字节总数	**100**		

值得注意的是 COPR.0 的长度随三个长度参数的不同而不同，这三个参数是：Provider Name Length、Signature Length 和 Auxiliary Data Length。提供这些参数是为了允许提供附加的定制性能，这些性能是软件支持的、但需要由 COPR.0 文件中的数据激活。

服务数据文件

在应用的示范中，我们在每个用户 SHA iButton 中都建立了一个服务文件 DLSM.102，并把它作为用户 iButton 识别。文件 DLSM.102 包含如下数据：

服务文件数据结构　表 2

服务文件 DLSM.102 的结构			
	字节数	抽样数据	备注
数据类型码	1	0	该域可用于进一步说明交易数据的类型。例如：0 表示动态数据，1 表示静态数据等等。
服务数据签名	20	[计算值]	由本地主机计算。
乘法器/转换系数	2	8B48H	该数据可以用来把一个数值转换为另一个数值。在本服务示范中，我们安装了美元的国际流通代码和从美分转换为美元的转换系数 0.01。特殊格式通常由服务确定。
账目余额	3	01 86 A0H	100000 分 （$1000）
交易 ID	2	1234H	交易 ID 可以将更多的参考数据与交易链接到一起。
字节总数	**28**		

值得注意的是文件内容有 28 个字节，但实际要以下面的格式存储 32 个字节（更多的信息参见 AN114）：

文件长度 (1 字节)	文件内容 (28 字节)	文件延续指针 (1 字节)	CRC-16 (2 字节)

本地主机检测用户 SHA iButton 时，它从用户 SHA iButton 的目录列表中读取文件 DLSM.102 的页码（在我们的服务示范中是 13），使用与第 13 页相关的密钥（密钥为 5）进行器件认证。

安装服务数据

在处理任何交易之前，必须把适当的服务数据和密钥安装到协处理器和用户 iButton 中 — 也可以看作是器件的初始化过程。静态和动态数据服务都需要器件认证。如果服务采用动态数据（如电子付费应用），需要在每次交易中验证和更新服务数据。协处理器将包含两个系统密钥：一个用于认证，另一个用于验证服务数据、并对服务数据再次签名。除非系统签名密钥一定安装到 0 号密钥中，安装两个系统密钥的步骤是相同的，通常使用不同的输入数据（片断）计算。在用户器件端，通过在服务数据页嵌入一个由服务数据和系统签名密钥计算产生的签名，可以保护服务数据。把服务安装到协处理器和用户 SHA iButton 中的基本步骤如下所述，以下给出了详细的说明。

对不需要数据验证的服务：

（1） 把系统认证密钥安装到协处理器中
（2） 通过将系统认证密钥绑定到用户 iButton 地址编码和服务数据页码，在用户 iButton 中安装一个唯一的器件认证密钥
（3） 向用户 iButton 写服务数据（如果需要的话）

对需要数据验证的服务：

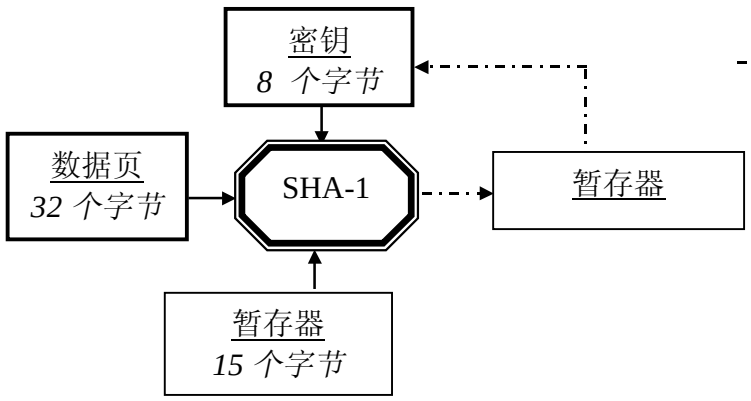
（1） 把系统认证密钥安装到协处理器中
（2） 把系统签名密钥安装到协处理器中
（3） 通过将系统认证密钥绑定到用户 iButton 地址编码和服务数据页码，在用户 iButton 中安装一个唯一的器件认证密钥
（4） 利用系统签名密钥、服务数据、服务数据的页码和写次数计数值、以及用户器件地址编码计算签名
（5） 在服务数据中嵌入签名，并把它写入用户 iButton

SHA iButton 中密钥的产生

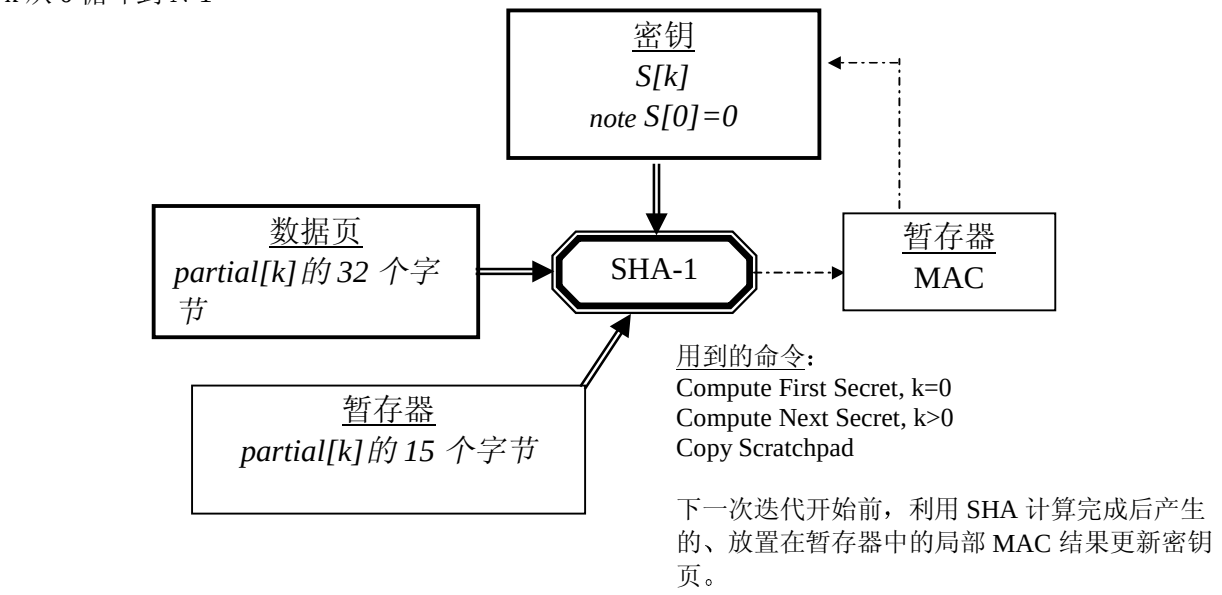
把密钥安装到 SHA iButton 中需要把数据（片断，更多的信息参见 AN152）写到指定页和暂存器，用数据计算 MAC，把选定的 MAC 字节复制到目标密钥内（参见图 4）。不要直接写入密钥。通过片内 SHA 引擎产生的密钥允许几个用户共同参与密钥的安装过程（称为密钥分摊），如果没有其它用户的参与，任何单个用户都不能再生系统密钥。这个过程有效地减小了密钥泄漏的风险。从 *N* 个片断（*partials[k]，k=0 至 N-1*）产生系统密钥的流程参见图 5。每次迭代计算 MAC 时利用两个数据源：47 个输入数据字节（其中 32 个字节写入数据页，15 个字节写入暂存器）和密钥的当前内容（8 个字节）。密钥用下次迭代之前刚刚计算出的新的 MAC 进行更新。循环的初始（*k=0*）密钥设置为一个空值。因为每一次迭代都为下一次计算产生一个新的密钥，所以，最后的系统密钥是上述所有输入片断的函数。

除系统签名密钥必须安装到协处理器的 0 号密钥（即 *cSignSecretNum=0*）以外，上述流程对系统鉴别密钥和系统签名密钥的计算同样适用。执行通用函数：*installSystemSecret* 可以把来自多个片断的系统密钥安装到 SHA iButton 中，详细内容见第 0 章。

SHA iButton 中产生的密钥　图 4



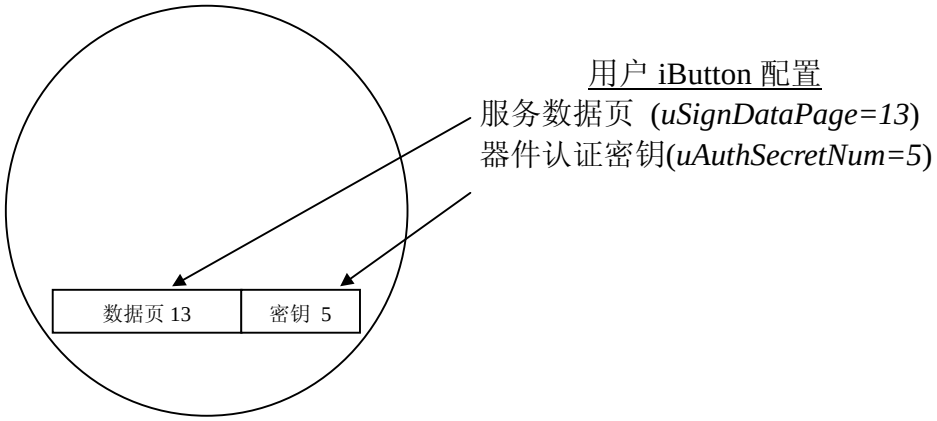
使用密钥共享产生系统密钥　图 5



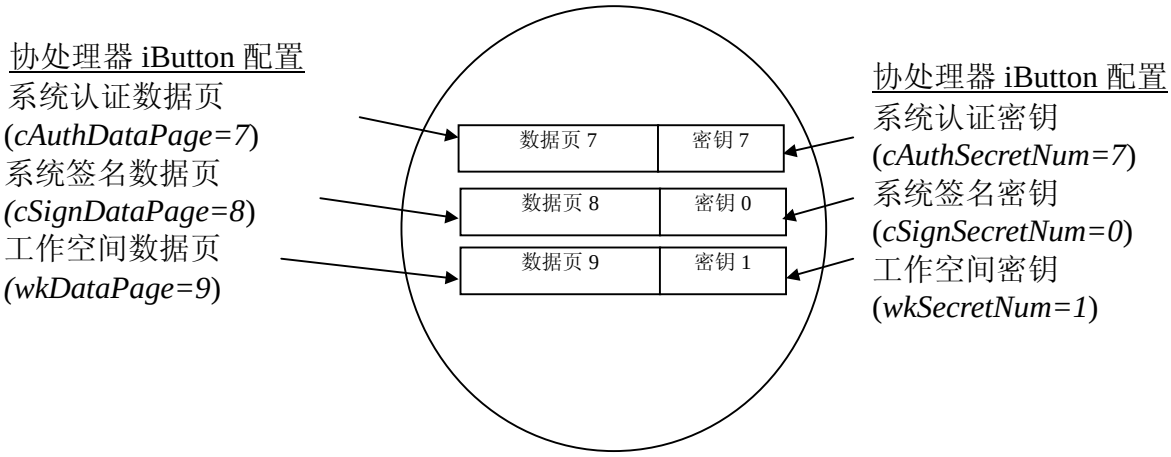
服务示范

我们的服务示范是将账目余额计入借方的一次电子付费服务，每次交易都要重新签名。值得注意的是器件认证不仅要验证服务数据页，而且还包括与用户 iButton 中服务数据页有关的器件认证密钥。对于服务数据签名，系统的一般签名密钥存在协处理器中，而不是存在用户 iButton 中。因此，我们可以只用一个数据页和一个密钥来控制电子付费应用：服务数据页和与之相匹配的器件认证密钥。本服务示范中用到的符号和变量概述如下以供参考。值得注意的是在多服务寄主器件中，每个用户 iButton 的服务数据页码（*即 uSignDataPage*）和认证密钥号（*uAuthSecretNum*）可以不同。

在用户 SHA iButton 中用到的页如下：



协处理器 SHA iButton 中用到的页如下：



变量和数值列表　表 3

变量列表		
变量	**服务示范取值**	**说明**
cAN		协处理器 iButton 地址编码
uAN		用户 iButton 地址编码
cAuthDataPage	7	系统认证数据页，在协处理器中
cAuthSecretNum	7	系统认证密钥号，在协处理器中
cSignDataPage	8	系统签名数据页，在协处理器中
cSignSecretNum	0	系统签名密钥号，在协处理器中
uAuthDataPage	13	用户器件认证数据页
uAuthSecretNum	5	用户器件认证密钥号
uSignDataPage	13	用户 iButton 中的服务数据页码
svcName	Dallas Semiconductor	器件供应商的名字，需要的时候在右侧填补字节 00H

变量列表		
变量	**服务示范取值**	**说明**
nAuthPartials	1	计算系统认证密钥的片断的数量
nSignPartials	1	计算系统签名密钥的片断的数量
authPartial[j] *j=0 至 nAuthPartials-1*	47FFH 字节	计算系统认证密钥的片断—每个矩阵元素中有 47 个字节：其中 32 个字节写到数据页中，另外 15 个字节写到暂存器中
signPartial[j] *j=0 至 nSignPartials-1*	47FFH 字节	计算系统签名密钥的片断—每个矩阵元素中有 47 个字节：其中 32 个字节写到数据页中，另 15 个字节写到暂存器中
authBind	39 00H 字节	器件认证密钥的绑定数据—39 个字节的数据块，用于把系统认证密钥绑定到用户器件。前 32 个字节写入数据页，剩下的 7 个字节写入暂存器以计算器件认证密钥。
signCode	3 00H 字节	用于产生服务数据签名的 3 个字节代码
signInitial	20 00H 字节	计算服务数据签名的初始值
uData		服务数据页内容
TA1, TA2, ES		器件地址和状态寄存器
uSignDataPageWCC		服务数据页的写次数计数，在用户 iButton 中 *(=uAuthDataPageWCC*)
cFileName	COPR.0	协处理器文件名，存储在协处理器中
uFileName	DLSM.102	服务文件名，存储在用户 iButton 中
wkDataPage	9	协处理器中工作空间数据页码
wkSecretNum	1	工作空间密钥号，在协处理器中

在协处理器中安装系统认证密钥

下面列出了在 SHA iButton 中安装系统鉴别密钥的命令时序和数据流。执行 API 函数（*installSystemSecret*）的细节问题见附录 A。

在协处理器中安装系统鉴别密钥　　图 6

在协处理器中安装系统认证密钥

k 从 0 循环到 nAuthPartials-1

- 把 authPartial[k]的 32 个字节写到暂存器中 [Write Scratchpad]
- 把暂存器中的数据复制到 cAuthDataPage 页[Copy Scratchpad]
- 把 authPartial[k]的其余 15 个字节写到暂存器[Write Scratchpad]

❶

- 计算密钥；MAC 置于暂存器中
  [Compute First Secret]，如果 k = 0
  [Compute Next Secret]，如果 k > 0

❷

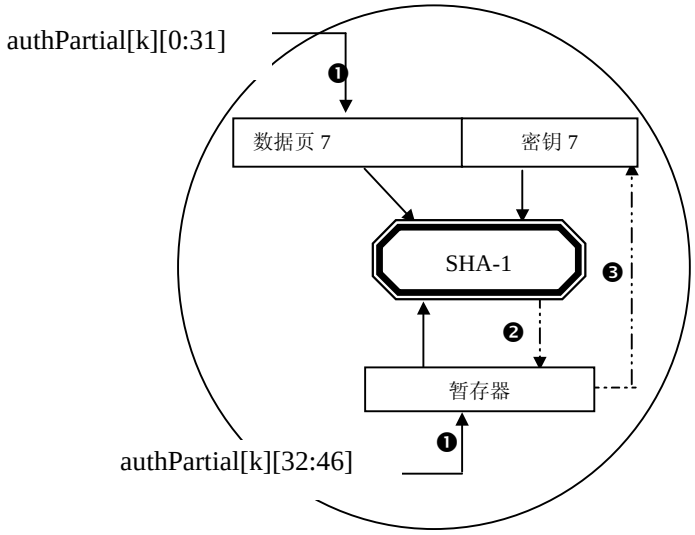
- 向暂存器写入空字节，准备复制密钥
  使用目标密钥地址[Write Scratchpad]
- 从暂存器把密钥复制到 cAuthSecretPage 密钥 [Copy Scratchpad]

❸

结束循环

为了在协处理器中安装系统认证密钥，要用到 API：
s=installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)
s=eraseDataPage(cAN,cAuthDataPage)

调用 *eraseDataPage*（具体实现在第 0 节）将删除最后一个写到协处理器中的片断，防止其泄漏。



协处理器
系统认证密钥被安装到协处理器 *cAuthSecretNum* (=7) 的密钥中。片断被写到 *cAuthDataPage* (=7) 页和暂存器，用于计算密钥。

请注意，为简单起见本文的 API 调用列表忽略了所有的误码校验和重复迭代。在实际应用中，每次调用返回的状态都应进行误码校验，并进行适当的重复迭代。例如，在协处理器中安装具有适当误码校验和重试功能的系统认证密钥的程序类似于以下代码：

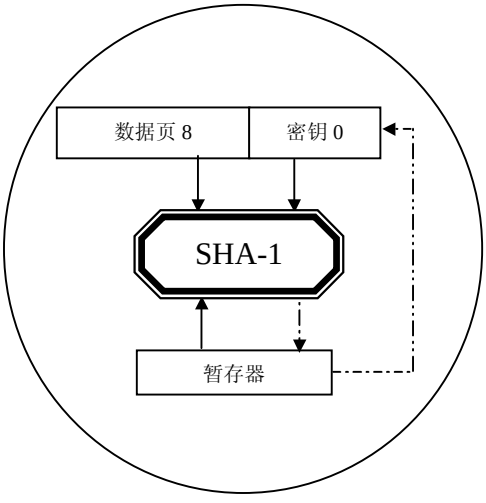
// iteration count limit
loopLimit = 5

```
s=1
loop=0
do while (loop<loopLimit and s<>0)
    // the call returns 0 if no error
    s=installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)
    loop=loop+1
end loop

// quit if we still have errors after preset number of tries
if(s<>0) then
        exit
end if

s=1
loop=0
do while (loop<loopLimit and s<>0)
    // the call returns 0 if no error
    s=eraseDataPage(cAN,cAuthDataPage)
    loop=loop+1
end loop

// quit if we still have errors after preset number of iterations
if(s<>0) then
        exit
end if
…continue …
```

在协处理器中安装系统签名密钥

在 SHA iButton 中安装系统签名密钥的命令时序和数据流与安装系统认证密钥的过程一样，只是片断、目标数据页和目标密钥（系统签名密钥必须装到密钥 0 中）不同。

为了在协处理器中安装系统签名密钥，用到了 API：
s=installSystemSecret(cAN,cSignDataPage, cSignSecretNum, nSignPartials,signPartial)
s=eraseDataPage(cAN,cSignDataPage)

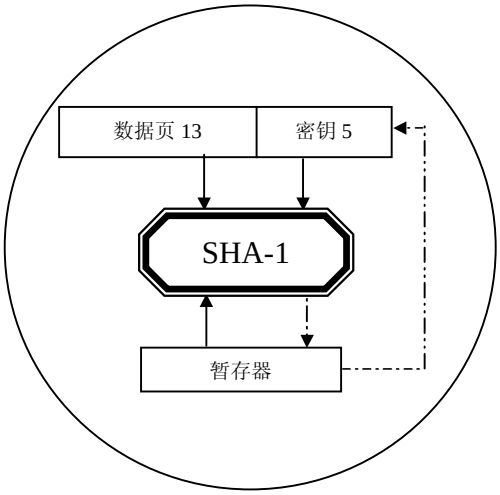


协处理器
系统签名密钥安装到协处理器的 *cSignSecretNum* (=0) 密钥中。
片断写到 *cSignDataPage* (=8) 页和暂存器中用来计算密钥。

值得注意的是系统签名密钥必须安装到密钥 0（*cSignSecretNum=0*）中，与它相匹配的数据页 *cSignDataPage* 或者是 0 或者是 8。在我们的服务示范中，第 0 页用作 OWFS 目录，所以我们用 *cSignDataPage* =8。

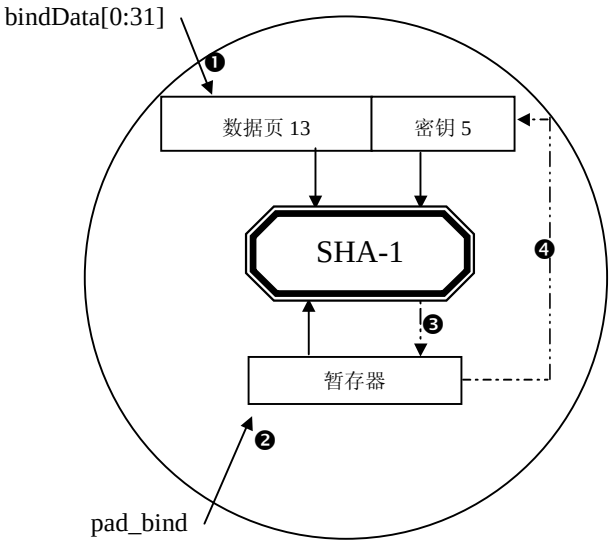
在用户 iButton 中安装器件认证密钥

为了安装用户 iButton 的器件认证密钥，首先要把系统认证密钥安装到用户 iButton 中（*secretAuthSystem*），然后由系统绑定数据（*bindData*）、系统认证密钥（*secretAuthSystem*）、服务数据页码（*uSignDataPage*）和用户 iButton 的地址编码（*uAN*）计算出唯一的器件密钥。用户 iButton 的地址编码使得器件认证密钥对 iButton 来说是唯一的。下面列出了这两个步骤，第二步执行了（详细内容见第 0 节）一个 API（*bindSecretToiButton*）。



用户 iButton Auth Secret 步骤 1
系统认证密钥(*secretAuthSystem*) 安装到用户 iButton 的密钥中 *uAuthSecretNum* (=5)。片断写到 *uAuthDataPage* (=13) 页 和暂存器用来计算密钥。

绑定器件和服务数据以产生器件认证密钥



用户 iButton Auth Secret 步骤 2
通过绑定系统认证密钥 (*secretAuthSystem*) 与服务和用户 iButton 特定数据 (*uAuthDataPage, uAN*)，把器件认证密钥 (*secretAuthDevice*) 安装在用户 iButton 的密钥 *uAuthSecretNum* (=5) 中。

截取绑定数据，暂存器中的服务和用户器件数据可用来计算唯一的器件认证密钥（*pad_bind*）：

偏移位置	0:7	8:11	12:12	13:19	20:22	23:31
字节数	8 个字节	4 个字节	1 个字节	7 个字节	3 个字节	9 个字节
数据	00H	bindData[32:35]	uAuthDataPage	uAN[0:6]	bindData[36:38]	00H

在用户 iButton 中安装系统认证密钥

k 从 0 循环到 nAuthPartials-1

- 把 authPartial[k]的前 32 个字节写到暂存器[Write Scratchpad]
- 把数据从暂存器复制到页 uAuthDataPage [Copy Scratchpad]
- 把 authPartial[k]的其余 15 个字节写到暂存器 [Write Scratchpad]

- 计算密钥；结果 MAC 置于暂存器中
  如果 k = 0 [Compute First Secret]
  如果 k > 0 [Compute Next Secret]

- 使用目的密钥地址向暂存器写入空字节，准备复制密钥 [Write Scratchpad]
- 把密钥从暂存器复制到密钥 uAuthSecretNum [Copy Scratchpad]

结束循环

绑定器件和服务数据以产生唯一的器件认证密钥

- 把 bindData 的前 32 个字节写入暂存器 [Write Scratchpad]
- 把数据从暂存器复制到 uAuthDataPage 页[Copy Scratchpad]

❶

- 将截取后的绑定数据（pad_bind）写入暂存器[Write Scratchpad]，该数据由 bindData、鉴别数据页码（uAuthDataPage）、用户 iButton 地址编码（uAN，无 CRC 校验码）的后 7 个字节组成

❷

- 计算密钥[Compute Next Secret]；结果 MAC 放置到暂存器中

❸

- 使用目标密钥地址向暂存器写入空字节，准备复制密钥 [Write Scratchpad]
- 把密钥从暂存器复制到密钥 uAuthSecretNum [Copy Scratchpad]

❹

为了安装器件认证密钥使用的 API：

s=installSystemSecret(uAN,uAuthDataPage,uAuthSecretNum,nAuthPartials,authPartial)

s=bindSecretToiButton(uAN,uAuthDataPage,uAuthSecretNum,bindData,uAuthDataPage,uAN)

s=eraseDataPage(uAN,uAuthDataPage)

在协处理器中创建服务数据签名

服务数据签名是根据服务数据和系统签名密钥计算产生的。签名可以嵌入[4]到服务数据页，在每次交易过程中，由本地主机进行验证。例如，在我们的服务示范中，我们把签名嵌入如下服务数据页（OWFS 需要数据块长度、后续指针和 CRC-16 校验字节）：

在服务数据页嵌入数据签名

偏移位置	0:0	1:1	**2:21**	22:23	24:26	27:28	29:29	30:31
字节数	1 个字节	1 个字节	**20 个字节**	2 个字节	3 个字节	2 个字节	1 个字节	2 个字节
数据	长度值	数据类型	**签名**	转换系数	账务余额	传输 ID	后续指针	CRC-16

因为签名占据了服务数据页的一部分，所以为了计算签名，签名数据块必须初始化为一些已知值（*signInitial*）[5]。注意，为了防止服务数据（建立“金额”）被非法复制和再利用，应该用诸如器件地址编码（*uAN*）、服务数据页的页码（*uSignDataPage*）和写次数计数值（*uSignDataPageWCC*）等器件的特殊数据作为输入进行签名计算。如果禁止复制服务数据，把服务数据安装到具有写次数计数器的页中也很重要，该计数器在每次写操作时都会递增。产生服务数据签名的过程是在 API *createDataSignature* 中实现的，详细内容见第 0 节。

// create signature, the passed uData_ini has the signature block initialized to signInitial

sig = createDataSignature(cAN,cSignDataPage,cSignSecretNum,uData_ini,signCode, uAN,uSignDataPage,uSignDataPageWCC)

被截取的签名数据块（pad_signCode）

偏移量	0:7	8:11	12:18	19:19	20:22	23:31
字节数	填补 8 个字节	4 个字节	7 个字节	1 个字节	3 个字节	填补 9 个字节
数据	*8 个 00H 字节*	*uSignDataPageWCC+1*[6]	*uAN[0:7]*	*uSignDataPage*	*signCode*	*9 个 00H 字节*

[4] 注意签名还没有嵌入到服务数据页，它可以存储到单独的页中，这样可留出 32 个字节空间存储原始的服务数据。

[5] 如果签名没被嵌入到 32 个字节的服务数据页中，就不必进行初始化：
sig=createDataSignature(cAN,cSignDataPage,cSignSecretNum,uData,signCode,uAN,uSignDataPage, uSignDataPageWCC)

[6] 已签名的服务数据写入到服务数据页后，WCC+1 就成为其新的写次数计数值，因为每次向页中写入数据写次数计数器的值都会增 1（在第 8 到 15 页）。

在协处理器中产生服务数据签名

- 将 uData_ini 写入暂存器 [Write Scratchpad]
- 将数据从暂存器复制到页 cSignDataPage [Copy Scratchpad]

❶

- 将截取后的签名数据(pad_signCode)写入暂存器[Write Scratchpad]，该数据由 *signCode*、*uSignDataPageWCC*、*uAN（无 CRC 校验码）*和 *uSignDataPage* 组成

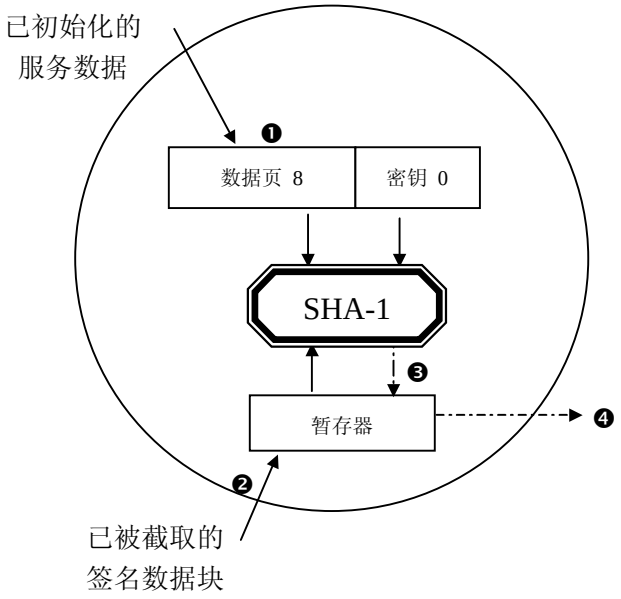
❷

- 计算签名 [Sign Data Page]；计算出的 MAC 置于暂存器

❸

- 从暂存器读取签名 MAC [Read Scratchpad]

❹



协处理器

服务数据签名根据以下数据计算而来：

- 初始化的服务数据(*uData_ini*)
- 系统签名密钥 (*cSignSecretNum*=0 时)
- 系统签名代码 (*signCode*)
- 用户 iButton 地址编码 *uAN* (无 CRC 校验码)
- 服务数据页码 (*uSignDataPage)* 和它的写次数计数值 *(uSignDataPageWCC)*

服务安装摘要

安装服务的主 API 调用概述如下。

无数据验证的静态数据服务

将服务数据安装到协处理器
// install the system authentication secret into the coprocessor
s=installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)

// erase the partial phrase so it is not accidentally exposed
s=eraseDataPage(cAN,cAuthDataPage)

将服务数据安装到用户 iButton
// install the system authentication secret into the user iButton
s=installSystemSecret(uAN,uAuthDataPage,uAuthSecretNum,nAuthPartials,authPartial)

// bind service data, user device ID with system secret to create a unique device auth secret
s=bindSecretToiButton(uAN,uAuthDataPage,uAuthSecretNum,bindData,uAuthDataPage,uAN)

s=eraseDataPage(uAN,uAuthDataPage) // erase the binding data from the data page

有数据验证的服务

将服务数据安装到协处理器
// install the system authentication secret into the coprocessor
s=installSystemSecret(cAN,cAuthDataPage, cAuthSecretNum, nAuthPartials,authPartial)

// install the system signing secret into the coprocessor
s=installSystemSecret(cAN,cSignDataPage, cSignSecretNum, nSignPartials,signPartial)

s=eraseDataPage(cAN,cSignDataPage) // erase the partial phrase so it is not accidentally exposed
s=eraseDataPage(cAN,cAuthDataPage)

将服务数据安装到用户 iButton
// install the system authentication secret into the user iButton
s=installSystemSecret(uAN,uAuthDataPage,uAuthSecretNum,nAuthPartials,authPartial)

// bind service data, user device ID with system secret to create a unique device auth secret
s=bindSecretToiButton(uAN,uAuthDataPage,uAuthSecretNum,bindData,uAuthDataPage,uAN)

// create the service data signature in coprocessor
sig=createDataSignature(cAN,cSignDataPage,cSignSecretNum,uData_ini,signCode,
uAN,uSignDataPage,uSignDataPageWCC)

// embed the signature in service data page
sData= …

// write the service data with the embedded signature to user iButton
s=writeDataPage(uAN,uSignDataPage,sData)

处理交易

用 SHA iButton 处理交易可以分成两步：本地主机认证用户 SHA iButton 和更新服务数据。本地主机含三个逻辑组件：协处理器、交易控制单元（TCU）和诸如显示器、令牌接受器及服务分配器的用户接口。TCU 可以是个人计算机或微控制器。重要的是协处理器和包括密钥、保密信息在内的其它单元要放置在适当的安全区域保证它们在物理上和电器上的安全性。

认证用户 iButton

图 7 说明了用户 SHA iButton 的认证过程，在 API 中的说明如下：

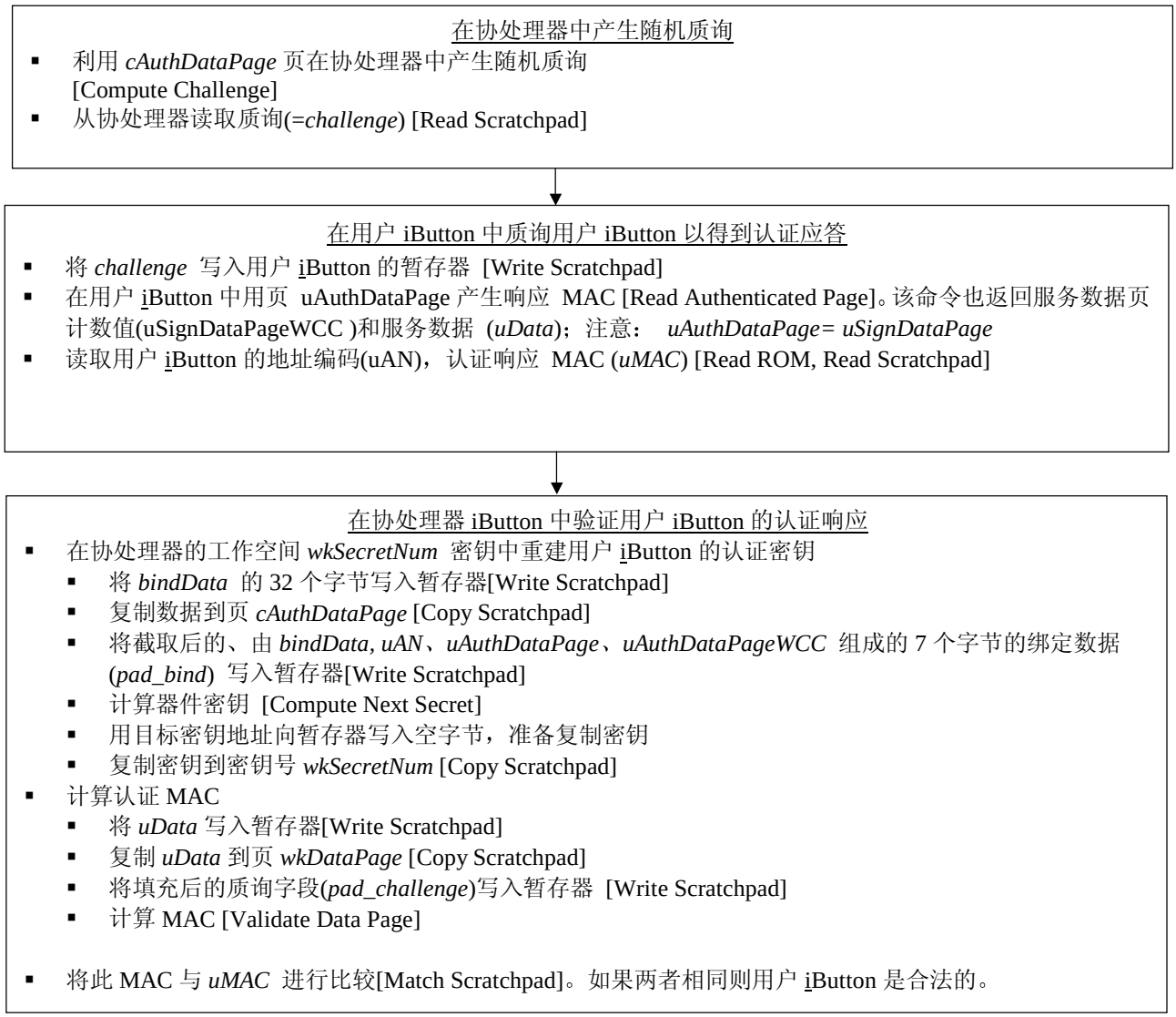
challenge=createChallenge(cAN,cAuthDataPage) // compute a challenge in the coprocessor

// challenge the user iButton for an authentication response
// the API call returns the service page data, page counter, and the response MAC
// uData=uResp[0:31]; uSignDataPageWCC=uResp[32:35]; uMAC=uResp[36:55]
uResp=answerChallenge(uAN,uAuthDataPage,challenge)

// recreate user iButton's device authentication secret in coprocessor
s=bindSecretToiButton(cAN,cAuthDataPage,wkSecretNum,bindData,uAuthDataPage,uAN)

// verify the response MAC in the coprocessor
s=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
uSignDataPageWCC,challenge,uMAC)

用户 SHA iButton 的认证过程　　图 7



截取的绑定数据块（pad_bind）

8 个字节	4 个字节	1 个字节	7 个字节	3 个字节	9 个字节
8 个 00H 字节	bindData[32:35]	uAuthDataPage	uAN[0:6]	bindData[36:38]	9 个 00H 字节

截取的质询数据块（pad_challenge）

填补 8 个字节	4 个字节	1 个字节	7 个字节	3 个字节	填补 9 个字节
8 个 00H 字节	*uAuthDataPageWCC*	*uAuthDataPage*	*uAN[0: 6]*	*challenge*	*9 个 00H 字节*

在协处理器中产生随机质询

为了在协处理器中产生一个随机质询，TCU 会令协处理器执行一次 Compute Challenge 操作。该命令完成后，TCU 可利用 SHA 引擎放置在暂存器中的 20 个 MAC 字节的任意 3 个字节。调用的 API 函数如下，具体执行细节见附录 A。

challenge = createChallenge(cAN, cAuthDataPage)

质询用户 iButton 以得到认证应答

TCU 向用户 iButton 发出质询（*challenge*），令其基于质询（写到暂存器中偏移量为 20 到 22 的单元）和它的器件认证密钥计算应答 MAC。实现的 API 见附录 A。

uResp=answerChallenge(uAN,uAuthDataPage,challenge)

返回的 uResp 含有服务数据（*uData*）、页写次数计数器（*uAuthDataPageWCC*）和应答 MAC（*uMAC*）。

uData=uResp[0:31]
uAuthDataPageWCC=uResp[32:35]
uMAC=uResp[36:55]

验证用户 iButton 的认证应答

为了验证用户 iButton 的响应，必须首先在协处理器中再生用户 iButton 的器件认证密钥。值得注意的是 TCU 不能读取这个再生的认证密钥，而且它也不需要去读取。认证不能通过直接比较认证密钥来实现，而是比较 SHA 根据认证密钥、其它服务和器件的特殊数据计算的结果。在协处理器中再生用户认证密钥后，执行 Validate Data Page 命令计算授权 MAC 用于比较。协处理器把它的计算结果（*cMAC*）存入外界不可读取的暂存器中。用 Match Scratchpad 命令比较从用户 iButton（*uMAC*）中读出的数值与协处理器暂存器中的 *cMAC*。

认证过程的第一步是在协处理器中再生用户器件的认证密钥，它在服务安装过程中与在用户 iButton 中安装器件认证密钥一样，只是这个认证过程的目标器件是协处理器、目标密钥安装到协处理器未用过的密钥编号区内。认证的第二步是执行用户 iButton 的应答 MAC（*uMAC*），见附录 A。

// recreate user device’s unique authentication secret in coprocessor – to an unused
// secret number (wkSecretNum)
s=bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)

// verify the user device’s response MAC in coprocessor
s=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
uAuthDataPageWCC,challenge,uMAC)

验证嵌入的服务数据签名

通过核对嵌入的签名（*sign*）验证服务数据页。首先，服务数据页的签名要被置为已知的初始值（*signInitial*），在协处理器中重新计算签名（*cSign*）。然后 TCU 比较 *sign* 与 *cSign*，如果二者一样，则服务数据是有效的。注意：在我们认证用户 iButton 时已经取得了服务数据、服务数据页的写次数计数值。

为了在协处理器中重建签名，首先要保存签名（*sign*）、然后用 *signInitial* 初始化 *uData* 签名数据以恢复产生签名的原始数据块（*uData_ini* ）。然后，我们利用用户器件和服务参数（地址编码、服务页码和计数值）来重建签名：

cSign=createDataSignature(cAN,cSignPage, cSignSecretNum, uData_ini, signCode, uSignDataPage, uAN, uSignDataPageWCC-1)

重新在更新的服务数据上签名

在动态数据服务中，服务数据被更新，计算一个新的签名用于服务数据的重新签名。这个过程与在用户 iButton 中安装初始服务数据的过程是一样的。如果 *uData_Update* 是更新后的服务数据，其签名数据被初始化为 *signInitial* ，那么新签名可以按如下方法产生：

sign=createDataSignature(cAN, cSignPage,cSignSecretNum, uData_Update, signCode,uSignDataPage, uAN, uSignDataPageWCC)

用 SHA iButton 处理交易的摘要

处理交易的主 API 调用总结如下。

无数据验证的静态数据服务

// create a random change code in coprocessor
challenge = createChallenge(cAN, cAuthDataPage)

// ask the user iButton to respond to the challenge
uResp=answerChallenge(uAN,uAuthDataPage,challenge)
uData=uResp[0:31]
uSignDataPageWCC=uResp[32:35]
uMAC=uResp[36:55]

// recreate user device's authentication secret in coprocessor
// in an unused secret number (wkSecretNum)
s=bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)

// verify the user device's response MAC in coprocessor
status=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage, uAuthDataPageWCC,challenge,uMAC)
…

有数据验证的静态数据服务

// create a random change code in coprocessor
challenge = createChallenge(cAN, cAuthDataPage)

// ask the user iButton to respond to the challenge
uResp=answerChallenge(uAN,uAuthDataPage,challenge)
uData=uResp[0:31]
using=uData[4:23]
uSignDataPageWCC=uResp[32:35]

uMAC=uResp[36:55]

// recreate user device's unique authentication secret in coprocessor
// in an unused secret number (wkSecretNum)
s=bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)

// verify the user device's response MAC in coprocessor
status=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage, uAuthDataPageWCC,challenge,uMAC)

// recreate the data signature in coprocessor
// initialize uData with *signInitial*
uData_ini=…
cSign=createDataSignature(cAN,cSignPage, cSignSecretNum, uData_ini, signCode, uSignDataPage, uAN, uSignDataPageWCC-1)

// compare the uSign and cSign
….

动态数据服务

// create a random change code in coprocessor
challenge = createChallenge(cAN, cAuthDataPage)

// ask the user iButton to respond to the challenge
uResp=answerChallenge(uAN,uAuthDataPage,challenge)
uData=uResp[0:31]
uSignDataPageWCC=uResp[32:35]
uMAC=uResp[36:55]

// recreate user device's unique authentication secret in coprocessor
// in an unused secret number (wkSecretNum)
s=bindSecretToiButton(cAN,cAuthDataPage, wkSecretNum, bindData, uAuthDataPage, uAN)

// verify the user device's response MAC in coprocessor
s=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage, uAuthDataPageWCC,challenge,uMAC)

// recreate the data signature in coprocessor
cSign=createDataSignature(cAN,cSignPage, cSignSecretNum, uData_ini, signCode, uSignDataPage, uAN, uSignDataPageWCC-1)

// compare the uSign and cSign
…

// update the service data and recompute signature
sign=createDataSignature(cAN, cSignPage,cSignSecretNum, uData_Update, signCode,uSignDataPage, uAN, uSignDataPageWCC)

// write the new service data and signature (*upData*) to the user iButton
s=writeDataPage(uAN,uAuthDataPage,upData)

// authenticate the user iButton and verify service data again
challenge = createChallenge(cAN, cAuthDataPage)

// ask the user iButton to respond to the challenge
uResp=answerChallenge(uAN,uAuthDataPage,challenge)
uData=uResp[0:31]
uSignDataPageWCC=uResp[32:35]
uMAC=uResp[36:55]

// make sure the user iButton received the updated data, compare uData and upData
…

// verify the user device's response MAC in coprocessor
status=verifyAuthResponse(cAN,wkDataPage,uData,uAN,uAuthDataPage,
uAuthDataPageWCC,challenge,uMAC)

附录 A: 执行 API

本节将讨论各种 API 的执行细节。开发人员应该能够将其应用于所选择的计算机语言中。注意这些列表不是实际可用的计算机代码。只是用它们来说明基本的命令和数据流时序。为简单起见，列表未考虑误码校验和重试循环。但在实际应用中，通常需要进行误码校验和重试过程。

基本的 1-Wire 器件 I/O 操作

假设应用于适当平台的驱动器已提供了必要的 1-Wire 器件访问函数。实际的函数名称可能与下列不同，但功能应该是一样的。我们假定下面的帮助函数已经提供给大家。

基本的 1-Wire I/O 操作 表 4

S=select(devAN)	选择地址编码(*devAN*)为给定参数的器件。如果器件选择成功，则调用的返回值为 0。
S=select()	选择 *devAN* 与前一次访问器件地址编码相同的器件。如果器件选择成功，则调用的返回值为 0。
S=resume()	恢复网络上前一次通信器件的通信。如果器件选择成功，则调用的返回值为 0。
S=reset()	复位 1-Wire 网，如果成功完成操作，则调用的返回值为 0。
Data=readBytes(len)	从选定器件读取字节值 *len*。
S=writeBytes(block,from,len)	向选定的器件写入 *len* 字节(从数据块 *block* 内偏移量 *from* 开始)。如果成功完成操作，则调用的返回值为 0。
TA1=lowAddress(page)	返回数据页码的低位地址字节。
TA2=highAddress(page)	返回数据页码的高位地址字节。
TA1S=lowSecretAddress(secretNum)	返回密钥码的低位地址字节。
TA2S=highSecretAddress(secretNum)	返回密钥码的高位地址字节。

C=CRC16(block,from,len,seed)	根据已知的种子值 *seed* 计算 *len* 数据单元的 CRC-16 校验码（数据块 *block* 内从偏移量 *from* 开始）。
C=invCRC16(block,from,len,seed)	根据传送的种子值 *seed* 计算 *len* 数据单元求反后的 CRC-16 校验码（数据块 *block* 内从偏移量 *from* 开始）。

向数据页写入数据 (WRITEDATAPAGE)

该 API 向数据页写入 32 个字节的数据块。

int status = writeDataPage(byte[] devAN, byte pageNum, byte[] data)

变量	字节	说明
devAN	byte[8]	目标 iButton 地址编码
pageNum	byte	目标页码
data	byte[32]	待写数据块
status	int	0 = 没有错误 状态值不为 0 意味着有错误发生。根据实际的执行情况，各状态值代表不同类型的错误。

TA1=lowAddress(pageNum)
TA2=highAddress(pageNum)

命令	数据流	注释
select(…)	devAN	选择目标 iButton 进行通信。
Erase Scratchpad	[W]: {C3H, TA1, TA2}	为后续 I/O 清除 HIDE 标志。
readBytes(len)		读取足够的（如 len = 5）状态字节检查操作是否完成。数值等于 AAH 意味着操作完成。
reset()		复位上面的操作。
resume()		与目标器件恢复通信。
Write Scratchpad	[W] {0F, TA1, TA2, data}	向暂存器写数据块。
readBytes(2)		读取上述数据流求反后的 CRC-16 校验码。TCU 应该计算同一数据流的 CRC-16 并和从目标器件读取的数值进行比较。不匹配意味着有 I/O 错误。
reset()		
resume()		
Read Scratchpad	[W] AAH [R] {TA1, TA2, ES}	读取地址和 ES 寄存器。 TCU 应该检查这些读取值是否正确。
reset()		
resume()		
Copy Scratchpad	[W] {55H, TA1, TA2, ES}	复制暂存器的数据到目标页。
readBytes(len)		检查操作状态。返回数据中 AAH 字节的值表明复制操作已经完成。
reset()		复位网络和调用返回值。

设置错误陷阱和重复预置循环数的简化方法是将上述步骤处于一个循环中，并在发生错误的时候使代码流跳至循环的开始部分。这种方法容易实现，但要牺牲其运行速度。更严谨的错误陷阱是在每次命令调用中检查错误、在有错误发生的时候，只重复冲突部分的调用。下表说明了这种简化方法。

命令	数据流	注释
loop=0 ***start:*** *loop=loop+1* *do while loop<=loopLimit*		
k=select(…)	devAN	选择目标 iButton 进行通信。
if(k<>0) goto start		
Erase Scratchpad	[W]: {C3H, TA1, TA2}	为后续 I/O 清除 HIDE 标志 。
readBytes(len)	{status bytes}	读取足够(如 len = 5)的状态字节以检查操作是否完成。数值 AAH 表明操作完成。
if (the last status byte is not AAH) goto start		
s1=reset()		复位上述操作。
s2=resume()		恢复与目标器件的通信。
if(s1<>0 or s2<>0) goto start		
Write Scratchpad	[W] {0F, TA1, TA2, data}	向暂存器写入数据块。
readBytes(2)	{inv CRC-16 bytes}	读取上述数据流求反后的 CRC-16 校验码。TCU 应该计算同一数据流的 CRC-16 校验码，并将其与读自目标器件的值进行比较。如果不匹配，则说明有 I/O 错误。
if (CRC bytes do not match) goto start		
s1=reset()		
s2=resume()		
if(s1<>0 or s2<>0) goto start		
Read Scratchpad	[W] AAH [R] {TA1, TA2, ES}	读取地址和 ES 寄存器。 TCU 应该检验这些读取值是否正确。
if (TA1, TA2, and ES do not match) goto start		
reset()		
resume()		
Copy Scratchpad	[W] {55H, TA1, TA2, ES}	把暂存器的数据复制到目标页。
readBytes(len)	{status bytes}	检查操作状态。返回数据中 AAH 字节的数值说明复制操作已经完成。
if (the last status byte <>AAH) goto start		
s1=reset()		复位网络。
end loop: *if(loop>loopLimit)* *status=1* *else* *status=0*		

删除数据页（ERASEDATAPAGE）

通过向数据页内填补 32 个字节的 FFH 删除其内容。

int status = eraseDataPage(byte[] devAN, byte pageNum)

变量	类型	说明
devAN	byte[8]	目标 iButton 地址编码
pageNum	byte	目标页码
status	int	参见上述关于返回 *status* 变量的说明

set data[32]={32FFH bytes}
status=writeDataPage(devAN, pageNum, data)

将 MAC 复制到密钥页（COPYMACTOSECRET）

API 复制暂存器中产生的 MAC 的 8 个字节到密钥地址存储器。这个函数通常在命令 Compute First Secret 和 Compute Next Secret 之后用于把暂存器中的部分 MAC 复制到密钥存储器。

int status = copyMACtoSecret(byte[] devAN, byte secretNum)

变量	类型	说明
devAN	byte[8]	目标 iButton 地址编码
secretNum	byte	目标密钥号

TA1S=lowSecretAddress(secretNum)
TA2S=highSecretAddress(secretNum)
set tmp_data[32]={32 00H bytes}

命令	数据流	注释
select(devAN)		选择目标 iButton 进行通信。
Write Scratchpad	[W] {0FH, TA1S, TA2S, tmp_data} [R] {inverted CRC-16 bytes}	为下一次复制操作建立地址标志。返回的 CRC 字节用于检测错误。
reset()		
resume()		
Read Scratchpad	[W] AAH [R] {TA1S,TA2S,ES}	读回地址和 E/S 寄存器。TCU 检查这些寄存器是否正确。
reset()		
resume()		
Copy Scratchpad	[W] {55H, TA1S, TA2S, ES}	将密钥从暂存器复制到密钥号。*（secretNum）* 注意暂存器的 8 个字节被复制到密钥存储器。
readBytes(len)		读取足够的状态字节以检测复制操作是否已经完成。
reset()		

安装系统密钥 (INSTALLSYSTEMSECRET)

在服务安装和处理交易的过程中，SHA iButton 安装系统密钥用到了协处理器和用户 iButton。安装系统认证密钥和安装系统签名密钥的过程是一样的，只是不同的片断、系统签名密钥必须要装到协处理器的 0 号密钥中。实现这一目的的 API 是 *installSystemSecret* 。

int status= installSystemSecret(byte[] devAN, byte pageNum, byte secretNum, int numPartials, byte[] partial)

变量名	类型	说明
devAN	byte[8]	目标 iButton 的地址编码
pageNum	byte	将片断写到器件的目标数据页
secretNum	byte	中间的和最后的目标密钥号
numPartials	int	片断的数量
partial	byte[size]	片断阵列，它被定义为一维阵列。*size=47*numPartials*

TA1=lowAddress(pageNum)
TA2=highAddress(pageNum)
TA1S=lowSecretAddress(secretNum)
TA2S=highSecretAddress(secretNum)

截取的片断 (*PAD_PARTIAL[J]*) 表 5

填补 8 个字节	*15 个字节*	*填补 9 个字节*
8 个 00H 字节	*partial[j][32:46]*	*9 个 00H 字节*

命令	**数据流**	**注释**
j 从 0 循环到 numPartials-1		
		将 partial[j] 写到数据页 pageNum。
writeDataPage(…)	devAN, pageNum, partial[j][0:31]	将 partial[j]的前 32 个字节写到数据页 pageNum。
		将填充后的部分*(pad_ partial)*写到暂存器。
resume()		
Write Scratchpad	[W] {0FH, TA1, TA2, pad_partial[j]} [R] {inverted CRC-16 bytes}	将填充后的绑定数据码写到暂存器。返回求反后的 CRC 字节用于误码检验。
reset()		
resume()		

If (j=0)
ComputeFirstSecret
else
ComputeNextSecret
	If (j=0)
	[W]{33H,TA1,TA2,0F}
	else
	[W]{33H,TA1,TA2,F0}
	
	[R] {inverted CRC-16 bytes}
		计算 MAC (secret)。
		根据 j 的值运行 Compute First Secret 或者 Compute Next Secret 函数。MAC 的部分结果置于暂存器中。

readBytes(len)		读取足够的（如 len=5） 状态字节以检测操作是否完成。值 AAH 说明成功。
reset()		
copyMACtoSecret(…)	devAN, secretNum	**复制 MAC 的 8 个字节到目标密钥。**
reset()		
循环结束		

将密钥绑定到用户 iButton (BINDSECRETTOIBUTTON)

通过将系统认证密钥绑定到服务数据页和器件地址编码产生唯一的器件认证密钥。

int status = bindSecretToiButton(byte[] devAN,byte pageNum,byte secretNum, byte[] bindData, byte uAuthDataPage, byte[] uAN)

变量	类型	说明
devAN	byte[8]	目标器件地址编码
pageNum	byte	目标数据页
secretNum	byte	认证密钥号
bindData	byte[39]	39 个字节数据块的系统通用绑定数据
uAuthDataPage	byte	绑定数据的数据页码
uAN	byte[8]	用户器件的地址编码

TA1=lowAddress(pageNum)
TA2=highAddress(pageNum)
TA1S=lowSecretAddress(secretNum)
TA2S=highSecretAddress(secretNum)

截取绑定数据(PAD_BIND) 表 6

填补 8 个字节	4 个字节	1 个字节	7 个字节	3 个字节	填补 9 个字节
8 个 00H 字节	*bindData[32:35]*	*uAuthDataPage*	*uAN[0:6]*	*bindData[36:38]*	9 个 00H 字节

命令	**数据流**	**注释**
writeDataPage(…)	devAN, pageNum, bindData[0:31]	**将绑定数据 bindData[0:31] 写到页 pageNum。**
		将截取后的绑定数据 (*pad_bind*) 写到暂存器。
resume()		
Write Scratchpad	[W] {0FH, TA1, TA2, pad_bind} [R] {CRC-16 bytes}	将截取后的绑定数据写到暂存器 TCU 检查返回的操作状态字节 CRC-16 以检测错误。
reset()		
		计算 MAC (device secret)。
resume()		
Compute Next Secret	[W]{33H,TA1,TA2,F0} [R] {bytes of inverted CRC-16}	运行 Compute Next Secret 函数。MAC 的部分结果存在暂存器中以进行下一次复制操作。
readBytes(len)		读取足够的状态字节以检测操作是否完成。
reset()		
copyMACtoSecret(…)	devAN, secretNum	**将器件密钥从暂存器复制到目标密钥。**

产生服务数据签名 (CREATEDATASIGNATURE)

该 API 利用服务数据、系统签名密钥、服务数据页码和它的写次数计数值、用户 iButton 的地址编码为服务数据计算签名。

byte sig[]=createDataSignature(byte[] devAN, byte pageNum, byte secretNum, byte[] uData, byte[] signCode, byte uSignDataPage, byte[] uAN, byte[] uSignDataPageWCC)

如果调用成功，则返回 20 个字节的 MAC。
产生服务数据签名用到的参数和数据列于下面：

变量	类型	说明
devAN	byte[8]	目标 iButton 的地址编码 (协处理器)
pageNum	byte	为计算签名，服务数据要被复制到的数据页码
secretNum	byte	存储系统签名密钥的密钥号（必须是 0）
uData	byte[32]	要签名的用户数据块
signCode	byte[3]	计算签名的系统通用代码
uSignDataPage	byte	用户 iButton 的服务数据页码，用于计算签名
uAN	byte[8]	计算签名的用户器件地址编码
uSignDataPageWCC	byte[4]	用户 iButton 中的写次数计数值(uSignDataPage)

TA1 = lowAddress(pageNum)
TA2=highAddress(pageNum)

考虑到 I/O 效应，*signCode* 和其它服务数据被截取成 32 个字节的数据块，并立即写入暂存器。

截取签名数据 (*PAD_SIGNCODE*) 表 7

填补 8 个字节	4 个字节	7 个字节	1 个字节	3 个字节	填补 9 个字节
8 个 00H 字节	*uSignDataPageWCC+1*[7]	*uAN[0:7]*	*uSignDataPage*	*signCode*	*9 个 00H 字节*

命令	**数据流**	**注释**
writeDataPage(…)	devAN, pageNum, uData	**将 uData 写入页 pageNum。**
		将截取后的签名数据写入 (pad_signCode) 暂存器。
resume()		
Write Scratchpad	[W] {0FH, TA1, TA2, pad_signCode} [R] {inv. CRC bytes}	将截取后的签名数据（pad_signCode）写入暂存器。
reset()		
		计算签名。
resume()		

[7] 签名后的服务数据写入服务数据页后， WCC＋1 就成为其新的写次数计数值，因为每次向页中写入数据，写次数计数值都将增加 1（在第 8 到第 15 页上）。

Sign Data Page	[W]{33H,TA1,TA2, C3} [R]{inv CRC-16 bytes}	执行命令 Sign Data Page。产生的 MAC 存入暂存器。
readBytes(len)		读取足够的状态字节以检测操作是否完成。
reset()		
resume()		
Read Scratchpad	[W] AAH [R]{TA1S, TA2S, ES, scratchpad data, and CRC bytes}	读取地址、ES 寄存器和暂存器。20 个字节的签名数据存储在偏移量从 8 开始的暂存器单元。
reset()		复位 1-Wire 网络和返回值。

产生质询字节 (CREATECHALLENGE)

API 返回 3 个字节的质询，该质询通常在协处理器中产生。SHA 计算把 SHA 计数值作为一个输入参数；每次执行 SHA 计算时，计数器都会递增，并且用户不会两次收到同一质询；从这个意义上说，该质询具有随机性。因为 SHA 计算从数据页和暂存器取得输入数据，所以把环境条件（如温度、湿度、噪声电平和探测器的力度等）作为计算的输入使得质询具有真正的随机性。执行 API 只对 SHA 计数器的值与存储页和暂存器中的内容做出应答，以提供不断变化的 MAC。

byte[] challenge=createChallenge(byte[] devAN, byte pageNum)

变量	类型	说明
devAN	byte[8]	目标 iButton 地址编码
pageNum	byte	目标数据页码。在这个简单执行中，只用到与该页有关的密钥。

TA1 = lowAddress(pageNum)
TA2 = highAddress(pageNum)

命令	**数据流**	**注释**
select(…)	devAN	选择进行通信的器件。
Erase Scratchpad	[W] {C3H, TA1, TA2}	清除 HIDE 标志以使暂存器数据可读。
reset()		
resume()		
Compute Challenge	[W]{33H,TA1,TA2,CCH}	执行 Compute Challenge 命令。
readBytes(2)	[R] {inverted CRC-16 bytes}	读取命令求反后的 CRC-16 的两个字节， TA1、 TA2 和控制字节。
readBytes(len)		检测 SHA 计算是否完成。值 AAH 表明成功。
reset()		
resume()		
Read Scratchpad	[R] {b0,b1,…b31}	从暂存器读取 32 个字节。 SHA 计算结果存储在偏移量为 8 到 27 的存储单元。选取任意的三个字节作为想要的质询 (=*challenge*)。
reset()		复位 1-Wire 网络并返回值。

应答认证质询 (ANSWERCHALLENGE)

当用户 iButton 收到本地主机质询的时候，用根据选定的数据页和器件认证密钥计算出的 MAC 进行应答。

byte[] resp=answerChallenge(byte[] devAN, byte pageNum, byte[] challenge)

如果调用成功，返回的一维字节阵列中包括以下数据，如果有错误发生则为空。

uData[32] = resp[0:31]	来自页 *pageNum* 的数据
pageWCC [4] = resp[32:35]	页 *pageNum* 的写次数计数器
uMAC[20] = resp[36:55]	认证应答 MAC

TA1=lowAddress(pageNum)
TA2=highAddress(pageNum)

变量名	类型	说明
devAN	byte[8]	目标器件地址编码
pageNum	byte	目标数据页码，注意该页码必须与器件认证密钥号匹配
challenge	byte[3]	质询字节
resp	byte[56]	一维数据阵列包括在上述的每个表中

考虑到 I/O 口的效应，质询字节通常被截取为 32 个字节块以便写入暂存器（*pad_challenge* ）：

截取质询字节 (PAD_CHALLENGE)

填补 20 个字节	3 个字节	填补 9 个字节
20 个 00H 字节	*challenge*	9 个 00H 字节

命令	**数据流**	**注释**
		将截取后的质询字节 (pad_challenge) 写入暂存器。
select(…)	devAN	选择用户信息扭进行通信。
Erase Scratchpad	[W]{C3H,TA1,TA2}	清除暂存器，使器件准备好接受数据。
reset()		
resume()		
Write Scratchpad	[W] {0FH, TA1, TA2, *pad_challenge*}	将截取后的数据写入暂存器。
reset()		
		计算应答 MAC。
resume()		
Read Auth. Page	[W] {A5H, TA1, TA2} [R] {32 bytes data from data page (*uData*), counter of data page (*pageWCC*), counter of secret page, inverted CRC-16 of command, address, data, and counter bytes}	(1) 该命令使得用户信息纽基于鉴别密钥、选定页的数据和来自暂存器的质询计算 MAC 。 (2) TCU 将读取数据页的内容、数据页的计数值、密钥页的计数值以及命令求反后的 CRC-16 校验码、地址、数据和计数器字节。
readBytes(len)		读取状态字节以检测 SHA 计算是否完成。
reset()		
		读取应答 MAC *(uMAC)*。
resume()		

Read Scratchpad	R: {TA1, TA2, ES, 32 bytes scratchpad data} uMAC starts at offset 8 and ends at 27	从暂存器读取计算结果。值得注意的是想要得到的 MAC (20 个字节)存储的起始偏移量为 8。
reset()		复位 1-Wire 网络和返回值。

验证认证应答 (VERIFYAUTHRESPONSE)

调用该 API 是为了验证用户 iButton 的鉴别应答是否正确。在协处理器的工作空间密钥中生成了用户 iButton 的器件认证密钥后必须调用该函数。

int status = verifyAuthResponse(byte[] devAN, byte wkDataPage, byte[] uData, byte[] uAN, byte uPageNum, byte[] uPageWCC, byte[] challenge, byte[] uMAC)

status = 0 用户应答 MAC (uMAC) 是有效的
1 用户应答是无效的
-1 有错误发生

变量名	类型	说明
devAN	byte[8]	目标器件地址编码
wkDataPage	byte	工作空间数据页码
uData	byte[32]	用户器件数据页（页码为 uPageNum）内容，在质询用户器件进行认证应答时获得
uAN	byte[8]	用户器件地址编码
uPageNum	byte	用户器件认证数据页码
uPageWCC	byte[4]	用户器件认证数据页写次数计数值
challenge	byte[3]	质询字节，用于质询用户 iButton 的认证 MAC
uMAC	byte[20]	用户 iButton 的认证 MAC

TA1W=lowAddress(wkDataPage)
TA2W=highAddress(wkDataPage)

考虑到 I/O 口效应，质询、其它服务和器件参数被截取为一个 32 个字节的页以便写入暂存器。

截取质询和服务参数 (PAD_AUTH)

填补 8 个字节	4 个字节	1 个字节	7 个字节	3 个字节	填补 9 个字节
8 个 00H 字节	*uPageWCC*	*uPageNum*	*uAN[0:6]*	*challenge*	*9 个 00H 字节*

命令	**数据流**	**注释**
writeToDataPage(...)	devAN,wkDataPage,uData	**将 uData 写入工作空间数据页。**
		向暂存器写入服务参数和质询字节。
resume()		
Write Scratchpad	[W] {0FH, TA1W, TA2W, *pad_auth*} [R] {inverted CRC-16 bytes}	将截取后的质询字节和服务参数写到暂存器。
reset()		
		计算应答 MAC。
resume()		
Validate Data Page	[W] {33H,TA1W, TA2W,3CH} [R] {inverted CRC-16}	计算结果置于暂存器中偏移量为 8 到 27 的存储单元。
readBytes(len)		读取足够的（如 len = 5）状态字节以检测操作是否完成。
reset()		
		核对用户 iButton 应答 MAC。
resume()		
Match Scratchpad	[W] {3CH, *uMAC*} [R] {inverted CRC-16 bytes, status bytes}	核对最后一个状态字节： AAH = 数据匹配 FFH = 数据不匹配
reset()		复位网络和返回值。