White Paper 1
SHA Devices Used in Small Cash Systems

www.maxim-ic.com

INTRODUCTION

Paying for products in the store is easy and convenient, provided that the store accepts "plastic money" and the convenience fee doesn't cut too deeply into the profit of the merchant. Naturally, for small purchases plastic is not accepted, which throws the customer back into the age of paper money and coins. Traditional vending machines operate on coins only. As prices for products and services increase the probability of having the right selection of coins on hand decreases. The situation isn't much improved by the bill changers, found on some machines, since they refuse bills that are worn out or damaged. In the worst case, one gets locked-in at a parking garage or gets fined for using public transit merely because of "monetary problems" at the point of sale. A way out of this dilemma is the introduction of small cash systems, that convert conventional money into "electronic cash" and store it in electronically readable tokens for spending at vending stations that are equipped to handle electronic cash.

The main components of such a system are as follows:

1) The electronically readable and writable token, which carries the electronic cash,
2) The commissioning station, which initializes new tokens and issues them,
3) The revaluing station, which converts money (cash, plastic) into electronic cash and transfers it into the token, and
4) The vending or POS station, which dispenses goods/tickets and deducts electronic cash from the token.

A small cash system is convenient and efficient whenever a limited group of people frequently uses services of the same organization. This could be at vending machines of a company cafeteria or private club, city-operated parking garages, local transit systems, fitness centers, or entertainment parks.

This document explains the major aspects of implementing a small cash system using Dallas Semiconductor iButton® products. In the two scenarios described in this document, the DS1963S serves as a cryptographic coprocessor as well as token. The DS1961S is suited as a token only. Other major components are the Secure Hash Algorithm (SHA-1) standard, microprocessor (host) hardware and firmware, the 1-Wire® protocol and secrets.

SHORT DESCRIPTION OF THE SHA-DEVICES INVOLVED

The DS1963S and the DS1961S are essentially memory iButtons with integrated 512-bit SHA-1 engine. Both devices support special command flows that feed data from various locations into the SHA-1 engine, start the engine and then make decisions based on the SHA-1 result and the data presented by a 1-Wire master. This master could be a microcontroller, which communicates with the iButton via the 1-Wire protocol. The DS1963S includes 16 pages of 32 bytes of NV SRAM, storage for eight 64-bit secrets, individual 32-bit write-cycle counters for 8 of the 16 memory pages and each secret, and a 32-bit pseudo-random number generator. For temporary storage and data verification, the DS1963S has a 32-byte scratchpad. The DS1961S includes only four pages of 32 bytes EEPROM and storage for one 64-bit secret, but no cycle counters or number generator. The size of the DS1961S's scratchpad is 8 bytes.

The 16 data memory pages of the DS1963S can be written just like any other Memory iButton. Writing to pages 8 to 15 increments the associated page write cycle counter, indicating that there was a change. Writing to the four memory pages of the DS1961S is only possible if one can make computations that involve the secret that is installed in the device. Reading the data memory of the DS1961S or DS1963S works the same way as with other Memory iButtons. There is no read-access to the secrets; they are "write-only".

The SHA-1 engine is a hardware implementation of the Secure Hash Algorithm, which produces a 160-bit output of one or more 512-bit sequentially processed input data blocks. Commonly used names for the SHA-1 output are "message digest", "signature" or "message authentication code" (MAC). The SHA-1 is called secure because it is computationally infeasible to find a message that corresponds to a given message digest, or to find two different messages that produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. The SHA-1 algorithm was chosen for its security and because it is an ISO/IEC standard (ISO10118-3).

The commands of the DS1963S that involve the SHA-1 engine are as follows:
- Read Authenticated Page
- Validate Data Page
- Sign Data Page
- Compute Challenge
- Authenticate Host
- Compute First Secret
- Compute Next Secret

This command set is designed to make the DS1963S suitable as multiple-service token, as a numeric coprocessor for use in vending stations and revaluing stations, as well as for user authentication for restricted log-in access to remote computers. The main difference between the SHA-1 commands consists in the composition of the data that is fed into the SHA-1 engine and the processing of the SHA-1 engine's output data. Except for Compute Challenge, all of the SHA-1 functions are applicable to small cash systems, as will be shown later in this document.

The commands of the DS1961S that involve the SHA-1 engine are as follows:
- Read Authenticated Page
- Copy Scratchpad
- Compute Next Secret

While Read Authenticated Page and Compute Next Secret operate essentially the same as with the DS1963S, the Copy Scratchpad command is very different from that used in other 1-Wire devices. In order to copy data from the scratchpad to the data memory of the DS1961S, one must transmit a message authentication code that has the device's secret as one of its input data components. As a consequence, there is no need to embed any signature in the service record ("electronic purse") that resides in a DS1961S. If the data made its way into the memory, it is authentic. With the DS1963S the situation is different. Since anybody can write data to the memory of a DS1963S, there is a need to embed a signature in the service record in order to verify its authenticity later. This difference between these devices affects the data format and the complexity of function flows at the various stations in the system. The other major factor is the smaller scratchpad size of the DS1961S, which requires four partial write and copy steps to write a full 32-byte memory page.

APPLICATION SCENARIOS

The DS1963S and DS1961S can be used to implement two different types of small cash systems. In scenario A, both the tokens and the coprocessor are DS1963S. In scenario B, the DS1963S functions as coprocessor in a system that uses the DS1961S as tokens or carriers of electronic cash.

A dual-mode system that uses both types of tokens is possible. Revaluing and vending station will need two DS1963S coprocessors. One coprocessor needs to be dedicated solely for DS1961S tokens and the other one solely for DS1963S tokens. Using a single DS1963S as a coprocessor in a dual-mode system would require a repeated reinstallation of the secret that is associated with memory page eight (the signing secret). During this procedure the data that the secret is composed of would be exposed to eavesdropping, compromising the system security and opening the door to an unauthorized refill of tokens with money.

The main advantage of scenario B (DS1961S token) is the lower cost of the tokens. Due to the message authentication code as part of the copy scratchpad protocol and the EEPROM write time, the communication of a debiting cycle takes longer than with a DS1963S token. The DS1961S is more vulnerable to power problems in a touch environment, since it has no internal energy source. Although, due to EEPROM technology, the number of write cycles that the DS1961S can achieve is lower than that of the DS1963S, this is not a considerable limit for practical use. The DS1961S can carry up to three service records or "purses" of a *single* service provider and is well suited for single-secret applications such as building access, cost control at copy machines, and paying at the cafeteria within a company, or for access to services at entertainment parks and private clubs.

The cost of a DS1963S is higher than a DS1961S. However, the DS1963S can accommodate up to seven service records of *different* service providers, which generates more flexibility. If all suitable memory pages are used, the price per service record of the DS1963S is much lower than that of a fully utilized DS1961S. Thanks to its internal energy source, with the DS1963S a copy scratchpad function will always complete, even if the electrical contact breaks. This is a clear advantage over the DS1961S particularly in environments where speed is critical. For these reasons, the DS1963S is the prime candidate for city-sized applications**,** such as local transit, parking garages, and independent local businesses that want to share the token.

Regardless of the type of token used in the system, the physical appearance of the commissioning, revaluing, and vending stations will be the same. Only the software and the format of the purse files (service records) are different.

DATA DETAILS

Regarding their data structure, the tokens are treated the same way as any other memory iButton. Data is organized in compliance with the 1-Wire File Structure. Using the 1-Wire File Structure, data is written in the form of files to the memory pages, similar to the operation of a floppy disk. This allows multiple files of different length and of different origin to reside in the same device. Using the 1-Wire File Structure, files can be added, modified or deleted without any conflict, as long as all parties that use the iButton strictly follow the same rules. The file structure specifies that the first page (page 0) of a memory iButton is reserved for the initial section of the device directory, which contains data management information and the entries of the first three data files that reside in the device. Table 1 shows the structure of a directory with one file entry and includes a short explanation of each field. The device directory looks the same for both types of tokens. For additional information on the file format see Dallas Semiconductor Application Note 114, *1-Wire File Structure.*

1-Wire Directory Structure With One File Entry Table 1

Offset	Field Name	Field Description
0	Length byte	**Format:** 1 byte, binary; value = 1Dh **Purpose**: requirement of the 1-Wire File Structure; precondition for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the CRC16 when reading iButton data files
1	Directory Mark	**Format:** 1 byte, binary; value = AAh **Purpose**: to allow identification of different iButton data formats **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: when formatting an iButton and to recognize whether and how an iButton is formatted
2	Map Address	**Format:** 1 byte, binary; value = 00h **Purpose**: requirement of the 1-Wire File Structure **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: when formatting an iButton
3	Bitmap Control	**Format:** 1 byte, binary; value = 80h **Purpose**: requirement of the 1-Wire File Structure **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: when formatting an iButton
4	Local Bitmap	**Format:** 4 bytes, binary **Purpose**: requirement of the 1-Wire File Structure; allows the operating system to find out whether and where there is room for more files or data. **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: when adding or expanding data files or deleting files
8	File name	**Format:** 4 bytes, ASCII capitals or numbers **Purpose**: requirement of the 1-Wire File Structure; to distinguish between different files within the same iButton **Defined by**: the application; see also Dallas Application Note 114 **Relevant**: to detect whether the iButton contains the file that is needed by the application
12	File Extension	**Format:** 1 byte, binary; value = 66h for purse files **Purpose**: requirement of the 1-Wire File Structure; to distinguish between different types of files **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to recognize whether a data file is a monetary purse
13	Start Page	**Format:** 1 byte, binary **Purpose**: requirement of the 1-Wire File Structure; to compute the physical starting address of the data file in the iButton memory **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to read a data file from the iButton memory
14	# Pages	**Format:** 1 byte, binary; value = 01h for purse files **Purpose**: requirement of the 1-Wire File Structure **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to know the length of a data file without reading all segments of a multipage data file
15	Continuation Pointer	**Format:** 1 byte, binary; value = 00h **Purpose**: requirement of the 1-Wire File Structure; allows iButton data files to be continued on any available memory page **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to find and read all segments of a multipage data file or directory
16	Inverted CRC-16	**Format:** 2 bytes, binary **Purpose**: requirement of the 1-Wire File Structure; for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the Length byte when reading iButton data files

The purse file format of the DS1963S is shown in Table 2. The length byte, continuation pointer, and CRC-16 work the same way as explained for the directory. The certificate type and algorithm field

identifies a particular data format within the purse file. Application Note 151 (see references) lists some of the formats that have been defined. The next field is the Service Data Signature. Since writing to the DS1963S is not restricted, the signature is the only means to verify the authenticity of the purse data. The Monetary Units Code and Multiplier field is based on the ISO-4217 standard, which assigns 3-digit decimal numbers to all existing currencies worldwide. A table of these codes can be downloaded from the University of British Columbia (see references). This standard does not define, however, how the currency codes are to be used in electronic media and whether the monetary value represents whole currency units (e.g., dollars) or fractions (cents). This decision is left to the software designer.

In software examples that are available from Dallas Semiconductor, the Monetary Unit Code and Multiplier (abbreviated "MUC") is constructed as follows: take the 3-digit decimal number that ISO-4217 has assigned to the currency and convert it into its binary equivalent. This determines the lower 10 bits of the MUC. The upper six bits indicate whether the monetary value needs to be divided or multiplied and what the divider or multiplier is, as shown in the following table.

Multiplier Code	000000	000001	000010	000011	100000	100001	100010	100011
Operation	× 1	× 10	× 100	× 1000	÷ 1	÷ 10	÷ 100	÷ 1000

Example: The ISO code for the US-dollar is 840 decimal or 1101001000 binary. To represent a value in US-dollars with a resolution of one cent (divide by 100), the resulting MUC is 100010 1101001000. Written in its hexadecimal form 8B48h, the MUC is then used in a purse file.

The monetary value or "balance" must be represented in compliance with the MUC. Continuing the example, a value of $12.34 is first converted into 1234 cents. Next the hexadecimal equivalent is computed, which is 4D2h. Since the balance field is three bytes long, leading zeros are added, which results in 0004D2h, the value that can be used in the purse file.

The Transaction ID is a random number that makes each transaction as it occurred at the vending or revaluing station unique. It is the precondition to prevent the so-called A-B-A attack, as described in *White Paper 3*. This same transaction number concept is frequently used for online credit card receipts, where the number appears as "authorization code".

DS1963S Purse File Format Table 2

Offset	Field Name	Field Description
0	Length byte	**Format:** 1 byte, binary; value = 1Dh **Purpose**: requirement of the 1-Wire File Structure; precondition for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the CRC16 when reading iButton data files
1	Certificate Type & Algorithm	**Format:** 1 byte, binary; value = 01h **Purpose**: to allow identification of different data formats in the token **Defined by**: Dallas Application Note 151 **Relevant**: when using more than one data format with the same type of token in the system.
2	Service Data Signature	**Format:** 20 bytes, binary **Purpose**: to verify authenticity of the service data (protect against fraud) **Defined by**: input data components are defined by Dallas Application Note 151; the algorithm for computing the 20-byte code is an ISO standard (SHA-1) **Relevant**: when checking service data for authenticity

Offset	Field Name	Field Description
22	Monetary Units Code & Multiplier	**Format:** 2 bytes, binary **Purpose**: to allow identification of different currencies and currency sub-units (e.. g., cents) **Defined by**: The code to identify the currency is defined by the ISO 4217 standard; see also Dallas Application Note 151 **Relevant**: when using more than a single currency in the system
24	Monetary Balance	**Format:** 3 bytes, binary number **Purpose**: to indicate the amount of money that is stored in the service record (purse) **Defined by**: the application, see also Dallas Application Note 151 **Relevant**: when adding to and when spending money from the electronic purse
27	Transaction ID	**Format:** 2 bytes, binary number **Purpose**: to assign a unique number to every transaction **Defined by**: the application **Relevant**: to prevent some types of possible attacks on the system
29	Continuation Pointer	**Format:** 1 byte, binary; value = 00h **Purpose**: requirement of the 1-Wire File Structure; allows iButton data files to be continued on any available memory page **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to find and read all segments of a multipage data file or directory
30	Inverted CRC-16	**Format:** 2 bytes, binary **Purpose**: requirement of the 1-Wire File Structure; for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the Length byte when reading iButton data files

The least significant byte of a multibyte field is stored at the lower address.

The purse file format of the DS1961S (Table 3) is very similar to the format used with the DS1963S. Since writing to the data memory of the DS1961S requires that one is able to compute the secret that is installed in the device, there is no need to include a signature inside the service data. The purse file, for this reason remains fairly short. Table 3 shows the details. A different certificate number indicates the different data format. The explanation of the other fields is the same as with the DS1963S purse file format.

DS1961S Purse File Format Table 3

Offset	Field Name	Field Description
0	Length byte	**Format:** 1 byte, binary; value = either 0Dh (if A segment is valid) or 15h (if B segment is valid) **Purpose**: requirement of the 1-Wire File Structure; precondition for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the CRC16 when reading iButton data files
1	Certificate Type & Algorithm	**Format:** 1 byte, binary; value = 03h **Purpose**: to allow identification of different data formats in the token **Defined by**: Dallas Application Note 151 **Relevant**: when using more than one data format with the same type of token in the system.
2	Monetary Units Code & Multiplier	**Format:** 2 bytes, binary **Purpose**: to allow identification of different currencies and currency sub-units (e.. g., cents) **Defined by**: The code to identify the currency is defined by the ISO 4217 standard; see also Dallas Application Note 151 **Relevant**: when using more than a single currency in the system

Offset	Field Name	Field Description
4	Dummy	**Format:** 4 bytes, binary; value = undefined **Purpose**: to make the monetary balance begin at a quarter-page boundary **Defined by**: Dallas Semiconductor **Relevant**: in conjunction with single-purse A-B scheme, which stores the current and the previous purse value in the purse file to safeguard against data loss due to electrical power problems in a touch environment
8	Monetary Balance of A segment	**Format:** 3 bytes, binary number **Purpose**: to indicate the amount of money that is stored in the service record (purse) **Defined by**: the application, see also Dallas Application Note 151 **Relevant**: when adding to and when spending money from the electronic purse
11	Transaction ID of A segment	**Format:** 2 bytes, binary number **Purpose**: to assign a unique number to every monetary transaction **Defined by**: the application **Relevant**: to prevent some types of possible attacks on the system
13	Continuation Pointer of A segment	**Format:** 1 byte, binary; value = 00h **Purpose**: requirement of the 1-Wire File Structure; allows iButton data files to be continued on any available memory page **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: to find and read all segments of a multipage data file or directory
14	Inverted CRC-16, valid if the length byte is 0Dh	**Format:** 2 bytes, binary **Purpose**: requirement of the 1-Wire File Structure; for data integrity check **Defined by**: 1-Wire File Structure; see Dallas Application Note 114 **Relevant**: in conjunction with the Length byte when reading iButton data files
16	Monetary Balance of B segment	(see A segment)
19	Transaction ID of B segment	(see A segment)
21	Continuation Pointer of B segment	(see A segment)
22	Inverted CRC-16, valid if the length byte is 15h	(see A segment)

The least significant byte of a multibyte field is stored at the lower address.

Table 3 lists two data segments (A and B) that contain monetary information. The B segment is introduced to increase data integrity in a touch-environment, where the electrical contact between token and reader/writer is unreliable. A purse file is installed with the A-segment only. With the next transaction, the B segment is added with a CRC16 that is computed over the higher (to be updated) length byte. In case that there was an electrical problem when writing the B segment, then the A segment is still valid, which is indicated by the low value of the length byte. Before the transaction is completed, the length byte needs to be updated to reflect the validity of the B segment. Vice versa, if the B segment was valid and the update of the A segment during another transaction fails, the data of the B segment is still intact, which is indicated by the high value of the length byte. This A-B scheme has been tested and was confirmed to be far superior in data integrity than using the A segment alone.

SCENARIO A: SECURITY CONCEPT (DS1963S Token)

In this scenario, the token can handle up to seven independent service records or purses, each with its individual Authentication Secret. Instead of using the same Master Authentication Secret (MAS) for the same purse in all tokens, each service record has its own Unique Authentication Secret (UAS). The MAS is computed through a SHA-1 operation from a 47-byte authentication input secret. The UAS is computed in another SHA-1 operation from the MAS, 32-byte binding data, page number of the purse or service record, and the seven bytes of the token's ROM ID.

Since there is unrestricted write access to the token, a signature is embedded in the purse file to verify its authenticity. The Master Signing Secret (MSS) required for generating and verifying the signature resides only in the coprocessors. The MSS is computed through a SHA-1 operation from a 47-byte signing input secret. For a more detailed description of all the parameters that are needed to compute the MAS and MSS see Table 4.

Knowing MAS, binding data, purse page number, and the token's ROM ID, the DS1963S coprocessor of any vending or revaluing station can compute the UAS of any purse that belongs to the system. Once the UAS is computed (assuming that the purse belongs to the system), one can

- verify whether that assumption was true, and if yes,
- verify the embedded signature of a purse
- generate a valid signature for a purse that belongs to the system

The verification of the purse involves comparing the SHA-1 result that the token has generated with a Read Authenticated Page command to a SHA-1 result that the coprocessor has computed in a Validate Data Page command. As a precaution against data replay attacks, the SHA-1 computation of the Read Authenticated Page command includes a 3-byte challenge that is taken from the scratchpad of the token. If this challenge consists of random data, the SHA-1 result will be different with every command execution, despite the fact that the token is the same and data inside the token has not changed.

To verify the embedded signature of a purse, the coprocessor first computes a SHA-1 result from the purse data, a 20-byte initial signature, the purse write-cycle counter, purse page number, seven bytes of the token's ROM ID and a 3-byte signing challenge. If the coprocessor got the same 20-byte result as is stored in the purse, the monetary value inside the purse is authentic.

SCENARIO A: ASSIGNMENTS AND PARAMETERS

A small cash system according to Scenario A uses DS1963S devices as coprocessors and as electronic tokens. A coprocessor is needed at the commissioning station, the revaluing station and at every vending/POS station. Three memory pages of the coprocessor are taken as

- Signing Page (coprocessor page 8, for technical reasons)
- Authentication Page (e. g., coprocessor page 9, arbitrary choice)
- Workspace Page (e. g., coprocessor page 10, arbitrary choice)

One of the token's memory pages is assigned as

- Token purse page (a page within range 9 to 15, defined when creating the purse file)

Scenario A uses several parameters or "system constants", as listed in Table 4. These parameters are typically stored as constants in the application firmware, which is loaded into a secure microcontroller and locked to prevent reading and disassembly.

Scenario A: Parameter List Table 4

Ref. #	Parameter Name	Description
SC32-1	Authentication Input Secret [0-31]	The 32-byte system constant which is loaded into the memory page to compute the initial Master Authentication Secret
SC15-1	Authentication Input Secret [32-46]	The 15-byte system constant which is loaded into the scratchpad of the coprocessor to compute the initial Master Authentication Secret.
SC32-S	Signing Input Secret [0-31]	The 32-byte system constant which is loaded into the signing page of the coprocessor before computing the Master Signing Secret (MSS).
SC15-S	Signing Input Secret [32-46]	The 15-byte system constant which is loaded into the scratchpad of the coprocessor before computing the Master Signing Secret (MSS).
SC32-B	Binding Data	The 32-byte system constant which is loaded into the token when binding the MAS to the token creating the Unique Authentication Secret (UAS). This constant is needed because the page data must be known.
SC7-B	Partial Binding Code	The 7-byte system constant which is loaded into the scratchpad of the token before computing the Unique Authentication Secret (UAS). SC7-B together with the purse page number and 7 bytes of the token's ROM ID are used together with SC32-B to compute the UAS.
SC20-S	Initial Signature	The 20-byte system constant, which is used in lieu of a real signature when computing the signature to be embedded in a purse file. This constant is needed because the signature must be known.
SC3-S	Signing Challenge	The 3-byte system constant, which is loaded in lieu of a real challenge into the scratchpad locations 20 to 22 before computing the signature to be embedded in a purse file. This constant is needed because the "challenge" must be known.

SCENARIO A: STEP DEFINITIONS

All activities that occur in this scenario can be described as individual steps that build on each other. There are "initialization steps" (marked by the letter I in the step names) and "field steps" (marked by the letter F). The Initialization steps are performed just once per service record or coprocessor and they occur in a controlled environment. The Field steps are typically performed in an uncontrolled environment, i.e., at the vending or revaluing station. Table 5 lists all of these steps together with their titles. For a detailed step description see *Appendix A*.

Scenario A: Step Definitions Table 5

Step Name	Step Description
AI1	Installation of the Master Signing Secret (MSS) in the coprocessor
AI2	Installation of initial Master Authentication Secret (MAS) in the coprocessor
AI3	Installation of the initial Master Authentication Secret (MAS) in the token
AI4	Installation of Unique Authentication Secret for the selected purse page
AI5	Installation of a device file directory with an entry for the purse file in the token
AI6	Writing a zero-value purse file to the token (compute signature, write data, verify)
AF1	Installation of the purse's UAS as the secret of the workspace page of the coprocessor
AF2	Verify whether the purse belongs to the system
AF3	Verify whether the purse data is valid
AF4	Updating the purse file in the token (compute signature, write data)
AF5	Verify whether the purse file was written successfully to the same token

SCENARIO A: COPROCESSOR SETUP

A DS1963S is required as coprocessor at all token commissioning stations and vending stations. In scenario A, the coprocessor needs to know two secrets, the Master Authentication Secret (MAS) and the Master Signing Secret (MSS). The MSS is installed first in step AI1 followed by the MAS in step AI2.

To increase the system security, the MAS should be built from several partial secrets. Ideally, partial secrets are installed by different people that only know their part of the secret. This way, if there is a security leak, only a partial secret will be exposed and the security of the system is not compromised. Working with partial secrets requires the repeated execution of step AI2. Coprocessors and tokens (in the process of being commissioned) must be exposed to all partial secrets that are used in the system in the same sequence. See section *Scenario A: Notes* for more details of this procedure. Since it is critical for determining the validity of the purse's data, the Master Signing Secret should also be built from partial secrets.

SCENARIO A: COMMISSIONING THE TOKEN

Before a DS1963S can function as a token, it needs to be "commissioned". This term combines all the steps that are necessary to make a purse in the token a part of "the system", i.e., the environment of which the token will become a member and in which it is supposed to work. Fresh from the factory, all the DS1963S are the same, except for their unique 64-bit ROM IDs. The commissioning procedure teaches them their Unique Authentication Secret (UAS) for the selected purse page, installs a device directory (i.e., the file name that the system uses for the purse) and creates a purse file without money. After this procedure, the device will be a token that is recognized as a member of the system, but it will not be good for buying anything, since the purse is empty. The detailed descriptions of the steps referenced in this section are found in *Appendix A*.

The commissioning of a token begins with the installation of the initial Master Authentication Secret of the system in the token (AI3) as the secret of the page that will later store the purse. This step is typically performed on a large number of tokens that are processed as a group or batch. If the system uses partial secrets, then an additional, modified AI3 step is necessary. See section *Scenario A: Notes* for more details.

Next the tokens need to learn their Unique authentication secrets and the purse files must be installed. The Unique Authentication Secret is installed in step **AI4** as the secret of the selected purse page. Step **AI5** installs the device directory with the purse file name. The final step **AI6**, installation of the purse file, requires the assistance of a coprocessor that has all the partial authentication secrets installed, i. e., a DS1963S that could as well be used in a vending or revaluing station. This step is fairly complex and re-uses functions that also occur in the field. Now the purse in the token is ready to carry money, which it receives at the revaluing station.

SCENARIO A: REVALUING THE TOKEN

Once a token is commissioned, its new purse file needs to be loaded with money at a revaluing station. This is the only place where real money gets involved, either in the form of a coin- or bill-accepting unit (off-line) or with an online connection to the banking network, just like a cash machine (ATM). Assuming that the system-specific purse file exists in the token directory and that the revaluing station knows how much money is to be added and that the station has access to that money, the following steps will take place to transfer the cash into the purse:

The coprocessor inside the revaluing station computes the UAS of the purse page and installs it as the secret of its workspace page (AF1). This enables the coprocessor to verify whether the purse belongs to the system. Next the token receives a challenge, performs a Read Authenticated Page command on the purse file, and generates a SHA-1 result, which the coprocessor then uses to verify whether the purse belongs to the system (AF2). If the purse is confirmed as a member of the system, the authenticity of the signature in the purse file is checked (AF3). If the signature is valid, a new signature for the updated purse is computed and the new purse data is written to the token (AF4). As a special security measure, the revaluing station then sends a new challenge to the token, reads the purse page again using the Read Authenticated Page command, verifies (as in step AF2) the authenticity of the purse and double-checks whether the value of the purse was updated (AF5). If everything worked correctly, the revaluing station prints a receipt and the whole transaction is finished in a fraction of a second.

SCENARIO A: PAYING WITH THE TOKEN

The token is of little use without vending stations that accept the token instead of cash. Regarding the processing of token data, the vending station and the revaluation station are almost exactly the same. By selecting the merchandise, the customer tells the machine how much money is to be subtracted from the purse. Next the token is presented to the machine where the following steps take place:

The vending station looks for the system-specific purse file in the token's directory. If the file is found, the coprocessor inside the vending station computes the UAS of the purse page and installs it as the secret of its workspace page (AF1). This enables the coprocessor to verify whether the purse belongs to the system. Next the token receives a challenge, performs a Read Authenticated Page command on the purse page, and generates a SHA-1 result, which the coprocessor then uses to verify whether the purse belongs to the system (AF2). If the purse is confirmed as a member of the system, the authenticity of the signature in the purse file is checked (AF3). If the signature is valid, and if the funds are sufficient for the purchase, the purse file in the token is updated (AF4). Before releasing the goods, the vending station sends a new challenge to the token, reads the purse page again using the Read Authenticated Page command, verifies (as in step AF2) the authenticity of the purse and double-checks whether the value of the purse was updated (AF5). If everything worked correctly, the customer gets access to the merchandise, which ends the transaction.

SCENARIO A: NOTES

The sections *Coprocessor Setup* and *Commissioning the Token* recommend the use of partial secrets when installing the Master Authentication Secret (see Application Note 152). Constructing the MAS from partial secrets is essentially the same as repeating steps AI2 and AI3 with different starting conditions. The use of partial secrets has no impact on AI4 or subsequent steps. There is no logical limit to the number of partial secrets that can be used in a system.

To work with two partial secrets (instead of a single one), these are the additional steps to be performed after the initial MAS is installed for the coprocessor (AI2) and the token (AI3):

> Define a second 47-byte Authentication Input Secret SC32-2, SC15-2 (see also Table 4. "-2" indicates that this is the second partial input secret; there could be a 3rd, 4th, etc.)

For the coprocessors to be deployed in the system:

> After step AI2, perform a modified AI2 (=AI2') with the following differences:
> Instead of SC32-1 use SC32-2.
> Instead of SC15-1 use SC15-2.

Instead of Compute SHA/1st Secret use Compute SHA/Next Secret.
All other specifics remain the same as in AI2.

For the tokens to be deployed in the system:

After step AI3, perform a modified AI3 (=AI3') with the following differences:
Instead of SC32-1 use SC32-2.
Instead of SC15-1 use SC15-2.
Instead of Compute SHA/1st Secret use Compute SHA/Next Secret.
All other specifics remain the same as in AI3.

The section *Coprocessor Setup* also recommends the use of partial secrets when creating the Master Signing Secret. Constructing the MSS from partial secrets is essentially the same as repeating step AI1 with different starting conditions. The use of partial secrets has no impact on AI2 or any subsequent steps. There is no logical limit to the number of partial secrets that can be used in a system.

To work with two partial secrets (instead of a single one), these are the additional steps to be performed after the initial MSS is installed for the coprocessor (AI1)

Define a second 47-byte Signing Input Secret SC32-S2, SC15-S2 (see also Table 4. "-S2" indicates that this is the second partial Signing Input Secret; there could be a 3rd, 4th, etc.)

For the coprocessors to be deployed in the system:

After step AI1, perform a modified AI1 (=AI1') with the following differences:
Instead of SC32-S use SC32-S2.
Instead of SC15-S use SC15-S2.
Instead of Compute SHA/1st Secret use Compute SHA/Next Secret.
All other specifics remain the same as in AI1. The method of installing the MSS in the coprocessors has no effect on the tokens.

Once tokens are deployed carrying a service record or purse file of a single service provider, the vacant memory pages (any of the unused pages in the range of page 9 to page 15) can be used for additional service records of other service providers. Page 8 should not be used for service records. See *Appendix C* for a detailed explanation. To install the secret of another service provider perform steps AI3 and AI4 for the selected page. The process of adding another file to the directory is similar to AI5. Instead of creating a new directory (which would eliminate the first service record), one simply adds another file entry to the existing directory. Writing the additional service record requires the same steps as writing the initial service record (AI6), only the memory page number and file content are different.

For prototyping purposes, the DS1963S function as a coprocessor can be emulated by the host processor. Revaluing stations or vending stations that use software instead of a coprocessor compromise the system security, since the secret could be recovered from disassembled program code of any such station in the system.

SCENARIO B: SECURITY CONCEPT (DS1961S Token)

This scenario is based on a single authentication secret. This limitation is set by the design of the DS1961S, which can only store one secret. Instead of using the same secret in all tokens, each token holds its own Unique Authentication Secret (UAS). The Master Authentication Secret (MAS) is computed through a SHA-1 operation from a 47-byte authentication input secret. The UAS is computed

in another SHA-1 operation from the MAS, 32-byte binding data, the binding page number and seven bytes of the token's ROM ID. For a more detailed description of these parameters see Table 6.

Knowing MAS, binding data, binding page number and the token's ROM ID, the DS1963S coprocessor of any vending or revaluing station can compute the UAS of any token that belongs to the system. Once the UAS is computed (assuming that the token belongs to the system), one can

- verify whether that assumption was true, and if yes,
- write to the token

The verification of the token involves comparing the SHA-1 result that the token has generated with a Read Authenticated Page command to a SHA-1 result that the coprocessor has computed in an Authenticate Host command. Writing to the token becomes possible by means of a SHA-1 result that the coprocessor computes in a Sign Data Page command before the copy scratchpad command is issued to the token. As a precaution against data replay attacks, the SHA-1 computation of the Read Authenticated Page command includes a 3-byte challenge that is taken from the scratchpad of the token. If this challenge consists of random data, the SHA-1 result will be different with every command execution, despite the fact that the token is the same and data inside the token has not changed.

SCENARIO B: ASSIGNMENTS AND PARAMETERS

A small cash system according to Scenario B uses a DS1963S as coprocessor and DS1961S devices as electronic tokens. A coprocessor is needed at the commissioning station, the revaluing station and at every vending/POS station. Three memory pages of the coprocessor are taken as

- Signing Page (coprocessor page 8, for technical reasons)
- Authentication Page (e. g., coprocessor page 9, arbitrary choice)
- Workspace Page (e. g., coprocessor page 10, arbitrary choice)

One of the token's memory pages is assigned as

- Token binding page for MAS and UAS installation (e. g., token page 1, arbitrary choice)

The binding page is later used for the primary purse. Secondary purses can be installed in pages 2 and 3; however, they share the same UAS.

Scenario B uses several parameters or "system constants", as listed in Table 6. These parameters are typically stored as constants in the application firmware, which is loaded into a secure microcontroller and locked to prevent reading and disassembly.

Scenario B: Parameter List Table 6

Ref. #	Parameter Name	Description
SC32-1	Authentication Input Secret [0-31]	The 32-byte system constant which is loaded into the memory page to compute the initial Master Authentication Secret
SC15-1	Authentication Input Secret [32-46]	The 15-byte system constant which is loaded into the scratchpad of the coprocessor to compute the initial Master Authentication Secret. The first four and the last three bytes of this constant must be FF because the token uses FF bytes in these locations when computing the next secret. The padding with 7 bytes FFh is needed to get the same SHA-1 input data as is used with the DS1961S.
SC8-1	Authentication Input Secret [36-43]	The 8-byte subset of SC15-1 which is loaded into the scratchpad of the token to compute the initial Master Authentication Secret. This constant is identical to bytes 5 to 12 of SC15-1.

Ref. #	Parameter Name	Description
SC1	Binding Page Number	The token page number that is used when installing the Master Authentication Secret (MAS) in the token and when binding the MAS to the token creating the Unique Authentication Secret (UAS).
SC32-B	Binding Data	The 32-byte system constant which is loaded into the token when binding the MAS to the token creating the Unique Authentication Secret (UAS). This constant is needed because the page data must be known.

SCENARIO B: STEP DEFINITIONS

All activities that occur in this scenario can be described as individual steps that build on each other. There are "initialization steps" (marked by the letter I in the step names) and "field steps" (marked by the letter F). The Initialization steps are performed just once per token or coprocessor and they occur in a controlled environment. The Field steps are typically performed in an uncontrolled environment, i.e., at the vending or revaluing station. Table 7 lists all of these steps together with their titles. For a detailed step description see *Appendix B*.

Scenario B: Step Definitions Table 7

Step Name	Step Description
BI1	Installation of an all-zero secret for the signing page of the coprocessor
BI2	Installation of the initial Master Authentication Secret (MAS) in the token
BI3	Installation of initial Master Authentication Secret (MAS) in the coprocessor
BI4	Installation of Unique Authentication Secret in token
BI5	Installation of a device file directory with an entry for the purse file in the token (includes write verification)
BI6	Writing a zero-value purse file to the token (includes write verification)
BF1	Installation of the token's UAS as the secret of the workspace page and signing page of the coprocessor
BF2	Verify whether the token belongs to the system
BF3	Updating the purse file in the token
BF4	Verify whether the purse file was written successfully to the same token

SCENARIO B: COPROCESSOR SETUP

A DS1963S is required as coprocessor at all token commissioning stations and vending stations. In scenario B, the coprocessor needs to know just the Master Authentication Secret (MAS). This secret is installed in one step only. See *Appendix B, Step BI3*.

To increase the system security, the MAS should be built from several partial secrets. Ideally, partial secrets are installed by different people that only know their part of the secret (see Application Note 152). This way, if there is a security leak, only a partial secret will be exposed and the security of the system is not compromised. Working with partial secrets requires the repeated execution of step BI3. At any stage, coprocessors and tokens (in the process of being commissioned) must physically move together from one partial secret installation place to the next. See section *Scenario B: Notes* for more details of this procedure.

SCENARIO B: COMMISSIONING THE TOKEN

Before a DS1961S can function as a token, it needs to be "commissioned". This term combines all the steps that are necessary to make a token a part of "the system", i.e., the environment of which the token will become a member and in which it is supposed to work. Fresh from the factory, all the DS1961S are the same, except for their unique 64-bit ROM IDs. The commissioning procedure teaches them their Unique Authentication Secret (UAS), installs a device directory (i. e., the file name that the system uses for the purse) and creates a purse file without money. After this procedure, the device will be a token that is recognized as a member of the system, but it will not be good for buying anything, since the purse is empty. The detailed descriptions of the steps referenced in this section are found in *Appendix B*.

The commissioning of a token begins with the installation of an all-zero "secret" in an auxiliary DS1963S coprocessor (step BI1). This step is required just once and the coprocessor can be used for commissioning a large number of tokens afterwards. With the assistance of the auxiliary coprocessor (after step BI1), the initial Master Authentication Secret of the system is installed in the token (BI2). This step is typically performed on a large number of tokens that are processed as a group or batch. If the system uses partial secrets, then an additional loops through the steps BI2 and BI3 are needed. See section Scenario B: Notes for more details.

Next the tokens need to learn their Unique authentication secrets and the purse files must be installed. These steps require the assistance of a coprocessor that has all the partial authentication secrets installed, i.e., a DS1963S that could as well be used in a vending or revaluing station. The installation of the MAS in the tokens must also have been completed. The Unique Authentication Secret is installed in step BI4.

The final token commissioning steps are fairly complex; they re-use functions that also occur in the field. Step BI5 installs the device directory with the purse file name. The purse file itself is installed in step BI6, immediately after BI5. Now the purse in the token is ready to carry money, which it receives at the revaluing station.

SCENARIO B: REVALUING THE TOKEN

Once a token is commissioned, its purse file needs to be loaded with money at a revaluing station. This is the only place where real money gets involved, either in the form of a coin- or bill-accepting unit (off-line) or with an online connection to the banking network, just like a cash machine (ATM). Assuming that the system-specific purse file exists in the token directory and that the revaluing station knows how much money is to be added and that the station has access to that money, the following steps will take place to transfer the cash into the purse:

The coprocessor inside the revaluing station computes the token's UAS and installs it as the secret of its workspace page (BF1). This enables the coprocessor to verify whether the token belongs to the system. As a preparation for writing to the purse file, the coprocessor installs the same UAS as the secret of its signing page (BF1). Next the token receives a challenge, performs a Read Authenticated Page command on the purse page, and generates a SHA-1 result, which the coprocessor then uses to verify whether the token belongs to the system (BF2). If the token is confirmed as a member of the system, the purse file is updated (BF3). As a special security measure, the revaluing station then sends a new challenge to the token, reads the purse page again using the Read Authenticated Page command, verifies (as in step BF2) the authenticity of the token and double-checks whether the value of the purse was updated (BF4). If everything worked correctly, the revaluing station prints a receipt and the whole transaction is finished in a fraction of a second.

SCENARIO B: PAYING WITH THE TOKEN

The token is of little use without vending stations that accept the token instead of cash. Regarding the processing of token data, the vending station and the revaluation station are almost exactly the same. By selecting the merchandise, the customer tells the machine how much money is to be subtracted from the purse. Next the token is presented to the machine where the following steps take place:

The vending station looks for the system-specific purse file in the token's directory. If the file is found, the coprocessor inside the vending station computes the token's UAS and installs it as the secret of its workspace page (BF1). This enables the coprocessor to verify whether the token belongs to the system. As a preparation for writing to the purse file, the coprocessor installs the same UAS as the secret of its signing page (BF1). Next the token receives a challenge, performs a Read Authenticated Page command on the purse page, and generates a SHA-1 result, which the coprocessor then uses to verify whether the token belongs to the system (BF2). If the token is confirmed as a member of the system, and if the funds are sufficient for the purchase, the purse file is updated (BF3). Before releasing the goods, the vending station sends a new challenge to the token, reads the purse page again using the Read Authenticated Page command, verifies (as in step BF2) the authenticity of the token and double-checks whether the value of the purse was updated (BF4). If everything worked correctly, the customer gets access to the merchandise, which ends the transaction.

SCENARIO B: NOTES

The sections *Coprocessor Setup* and *Commissioning the Token* recommend the use of partial secrets when installing the Master Authentication Secret (see Application Note 152). Constructing the MAS from partial secrets is essentially the same as repeating steps BI2 and BI3 with different starting conditions. The use of partial secrets has no impact on BI4 or subsequent steps. There is no logical limit to the number of partial secrets that can be used in a system.

To work with two partial secrets (instead of a single one), these are the additional steps to be performed after the initial MAS is installed for the token (BI2) and the coprocessor (BI3):

> Define a second 47-byte Authentication Input Secret SC32-2, SC15-2, SC8-2 (see also Table 6; as with SC15-1 7 bytes of SC15-2 have to be FFh. SC8-2 is identical to bytes 5 to 12 of SC15-2.)

For the tokens to be deployed in the system:

The coprocessor used here must not yet have been subjected to BI3'.

> After step BI2, perform a modified BI2 (=BI2') with the following differences:
> Skip the step that writes eight 00 bytes to scratchpad.
> Skip Load First Secret
> Instead of SC32-1 use SC32-2.
> Instead of SC8-1 use SC8-2.

All other specifics remain the same as in BI2.

For the coprocessors to be deployed in the system:

> After step BI3, perform a modified BI3 (=BI3') with the following differences:
> Instead of SC32-1 use SC32-2.
> Instead of SC15-1 use SC15-2.
> Instead of Compute SHA/1st Secret use Compute SHA/Next Secret.

All other specifics remain the same as in BI3.

Once tokens are deployed, the vacant memory pages can be used for additional purposes. The process of adding another file to the directory is similar to BI5. Instead of creating a new directory, one simply adds another file entry to the existing directory. Writing the additional file requires the same steps as writing a purse file (BI6), only the memory page number, file extension and file content are different.

A secondary application of the token could be for access control. Since the data in the token cannot easily be altered, it can store an access control file with a list of keywords that the lock checks against its own list. Connected with each keyword should be an expiration date that the lock compares to its own real-time clock. If one keyword matches and is not expired, access is granted. The electronic locks learn their keywords at the time of installation and store them in otherwise unused memory sections of the DS1963S inside the lock. Provisions should be made in the lock's firmware to read new keywords from specially formatted DS1961S tokens, which can be checked for authenticity by challenge and response. The lock should also maintain a list to store ROM IDs of lost tokens, to prevent unauthorized access before the keywords are expired.

By design of the chip inside the DS1961S, writing to the register page requires the knowledge of the secret that is stored in the chip. If the secret is directly written to the device or computed in one step, the secret is known and can be write-protected. If the secret is computed from partial secrets and if there are no security leaks, one does not know the secret. As a consequence, it is not possible to write to the register page, e.g., to write-protect the secret. Although this might look like a disadvantage, in reality it is the precondition for changing the secrets throughout a system and to recycle the tokens. This would not be possible if the secret were write-protected.

For prototyping purposes, the DS1963S function as a coprocessor can be emulated by the host processor. Revaluing stations or vending stations that use software instead of a coprocessor compromise the system security, since the secret could be recovered from disassembled program code of any such station in the system.

QUESTIONS AND ANSWERS

Are there things that I must not do to the token?

Yes, with a commissioned DS1961S token one must not use the commands Load First Secret and Compute Next Secret. Each of these commands will affect the secret stored in the device, making it impossible for the token to be recognized as part of the system. If the Master Authentication Secret of the DS1961S is not constructed from partial secrets, the secret can be write-protected. Similarly, with a commissioned DS1963S, one must not use the copy scratchpad command to install a new secret for an existing purse file or service record. The token would no longer be accepted at the vending or revaluing stations that use the particular purse file. If the signature in the purse file is computed as described in *Scenario A: Security Concept*, even rewriting a valid purse file (without any changes to the data) will make the purse file invalid, because writing increments the purse's page write cycle counter. These security features reveal tampering with the device or its data.

When designing a small cash system, what shall I do with the ROM IDs of the tokens and the secret write cycle counters (DS1963S)?

The token’s ROM ID is the key to reconstruct transaction history. For this reason, it should be used as the customer's account number. This even works if multiple providers share a single DS1963S token because they do not share databases. One could also append the purse page number to the ROM ID to make the account number unique. In a system that buses the DS1963S as tokens, the write cycle counter value of the secret that is associated with the purse page should be stored as well. A mismatch of the secret write

cycle counter in the token and in the account database could indicate a so-called competitor attack. See *White Paper 3* for more details.

What happens if I have messed-up the secret or the purse's page write cycle counter?
The extent of the trouble depends on the provisions that the service provider has taken to recover from such mishaps. If transactions are recorded using the token's ROM ID and transaction numbers, all the transaction history can be reconstructed and the secret and the lost value can be restored in the same or another token. The service provider may charge an administration fee for such recovery operations.

How can I see the balance of my electronic purse account?
The implementation of the balance view function depends on the service provider. The revaluing stations should be designed to display the current balance. Vending stations should be able to display it too, if you are not concerned that others can look into your electronic purse. More convenient online viewing, as it is known from home banking, is possible as well. However, there may be a significant delay between a transaction and its posting for online view, unless all vending and revaluing stations regularly dial in to the central server (or are called by the server) to report the day's transactions. If the service provider uses a purse format as described in this document or discloses the chosen format, then you can read the balance with the iButton Viewer, a program than can be downloaded for free from the iButton website.

Can I deposit money into my token through the Internet, just like home banking?
Technically, this is possible, but only if your token service provider supports home banking and revaluing from home and allows you to obtain (download) and install the necessary software. You also need to buy a 1-Wire adapter with a suitable probe for the token.

Does revaluing from home compromise the system security?
Not at all, because neither the authentication secret nor the signing secret is exposed at any given time. The security of the SHA-1 algorithm is so high that even recording the communication with the token will not deliver clues that could lead to discovering the secret. Even if the secret of one token were known, this would not affect the whole system, because the token secrets are made unique (UAS).

Is there any risk of accidentally deleting purse files or making a purse invalid when reading the token with the iButton Viewer?
As long as you stay away from functions that issue the commands Load First Secret and Compute Next Secret (DS1961S) or copy scratchpad to a purse file or purse secret (DS1963S) you are safe. If the secret of a DS1961S is write-protected, there is no risk to accidentally change the secret. In any case, read the iButton Viewer User's manual thoroughly to know exactly what you are doing. The viewer does not know whether the device you are evaluating contains any money.

How will the security be affected if I stay with the DS1963S purse file format, but use a DS1963L or DS1993L as a token?
The DS1963L has page write-cycle counters, just like the DS1963S. However, the DS1963L has no built-in SHA-1 engine or any secrets. Therefore, the authenticity of the device cannot be verified through Challenge and Response. This opens the door emulation attacks, as explained in *White Paper 3*. The DS1993L as a token is even weaker than the DS1963L. Since it has no page write cycle counters, one could make a copy of the purse file, and as soon as the money is spent, restore the full purse, without even needing an emulator.

I will move out of the area where I can pay with these electronic tokens, but I haven't spent all my electronic cash. Can I get a refund?
This depends on the policy of the service provider. They should have the ability to decommission a token, i.e., to empty the purse file and refund the unused portion of your money. The token itself remains usable. It can be recycled and given to another user without any special processing. The steps of decommissioning a token, as far as the token is concerned, are the same as at a vending station. The new purse value is zero after the decomissioning is completed. Decommissioning a token at a revaluing station is not recommended, since it would create an incentive to steal tokens and to convert them into cash.

How can I implement a dual-currency system?
There are two different ways to do this. One approach is to implement two purses, one for each currency. The other approach is to use a single purse for a single currency and convert the amount whenever a transaction in the other currency takes place. The two-purse approach is easier to implement, but has the disadvantage that one has to put money into two purses. If one purse is empty, one cannot automatically spend funds from the other purse. Filling purses with different currencies requires a dual-currency revaluing station or different revaluing stations, most likely in different countries (e.g., at border crossings). The one-purse approach is more convenient for the user. However, it requires smarter vending and revaluing stations that can download exchange rates regularly, e.g., every night. Exchange rate services are available on the Internet (see references).

Is there some ready-made software that I can start with?
Yes, there is, see references. Application Note 157 describes a SHA iButton API and pseudocode, which is equivalent to the steps of Appendix A. A DS1963S SHA 1-Wire API Users Guide is found in Application Note 156. Matching source code examples are included in the 1-Wire Public Domain Kit, which can be downloaded for free from the iButton web site.

How secure are the tokens against various types of attacks?
Security aspects are discussed in *Appendix C* of this document and in *White Paper 3*.

REFERENCES

Data Sheet DS1963S: http://pdfserv.maxim-ic.com/arpdf/DS1963S.pdf
Data Sheet DS1961S: http://pdfserv.maxim-ic.com/arpdf/DS1961S.pdf
SHA-1 Secure Hash Standard: http://www.itl.nist.gov/fipspubs/fip180-1.htm
ISO-4217 Currency Codes: http://pacific.commerce.ubc.ca/xr/currency_table.html
Currency Exchange Rates: http://www.oanda.com/
White Paper 3, *Why are 1-Wire SHA-1 Devices Secure?:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/wp3.pdf
White Paper 4, *Glossary of 1-Wire SHA-1 Terms:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/wp4.pdf
White Paper 8, *1-Wire SHA-1 Overview:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/WP8.pdf
App Note 114, *1-Wire File Structure:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/app114.pdf
App Note 151, *Dallas Digital Monetary Certificates:* http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=827
App Note 152, *SHA iButton Secrets and Challenges:* http://dbserv.maxim-ic.com/appnotes.cfm?appnote_number=835
App Note 156, *DS1963S SHA 1-Wire API Users Guide:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/app156.pdf
App Note 157, *SHA iButton 1-Wire API Overview:* http://pdfserv.maxim-ic.com/arpdf/AppNotes/app157.pdf
iButton Viewer Download: http://www.ibutton.com/software/tmex/index.html
1-Wire Public Domain Kit download page: http://www.ibutton.com/software/1wire/wirekit.html

APPENDIX A

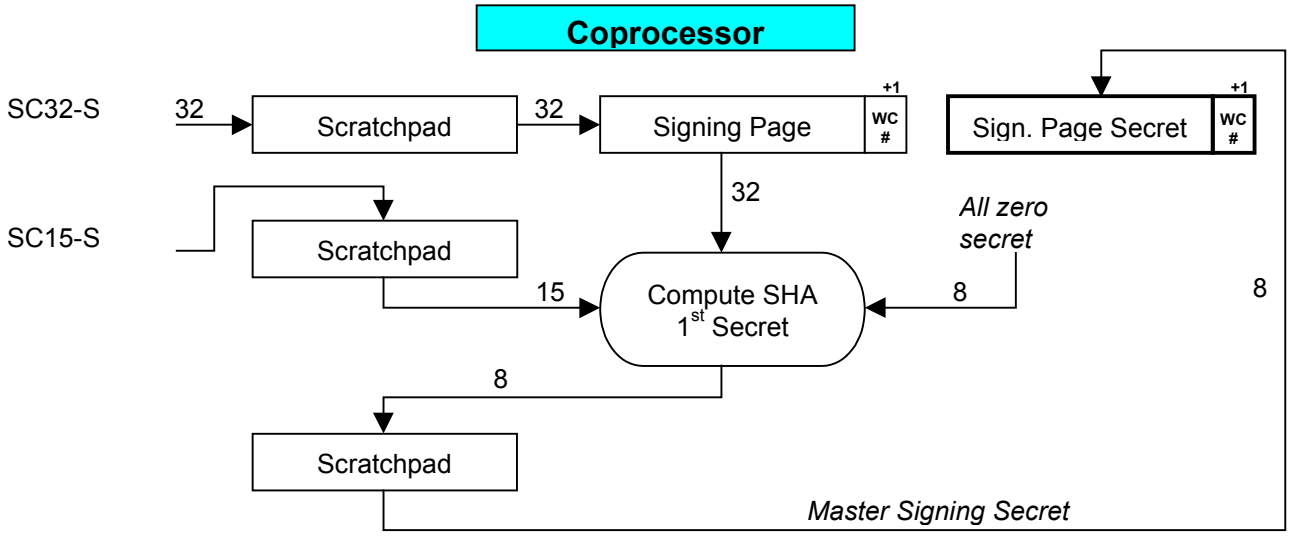
Step AI1

Title: Installation of the Master Signing Secret (MSS) in the coprocessor

Precondition:
- SC32-S is defined
- SC15-S is defined

Performed:
- When setting up the coprocessor for use in the system

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Using a dummy starting address, erase the scratchpad.	
2)	Using the starting address of the signing page, write SC32-S to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
3)	Fill scratchpad locations 8 to 22 with SC15-S.	
4)	Using the starting address of the signing page issue Compute SHA/1st Secret.	
5)	Using the starting address of the secret of the signing page, write 8 dummy bytes to the scratchpad.	
6)	Using the starting address of the secret of the signing page and a computed E/S byte, issue the Copy Scratchpad command.	
1)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
2)	To load the first 32 bytes of the Signing Input Secret into the Signing Page (page 8). *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the remaining 15 bytes of the Signing Input Secret into the scratchpad. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and an all-zero secret. *The target address must point to a location within the Signing Page.*
5)	To select the secret of the Signing Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Signing Page. At least 1 dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
6)	To make the SHA-1 result the secret of the Signing Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

DS1963S Secret Address Assignments:

Description	Address Range
Secret of Page 8	0200h to 0207h
Secret of Page 9	0208h to 020Fh
Secret of Page 10	0210h to 0217h
Secret of Page 11	0218h to 021Fh
Secret of Page 12	0220h to 0227h
Secret of Page 13	0228h to 022Fh
Secret of Page 14	0230h to 0237h
Secret of Page 15	0238h to 023Fh

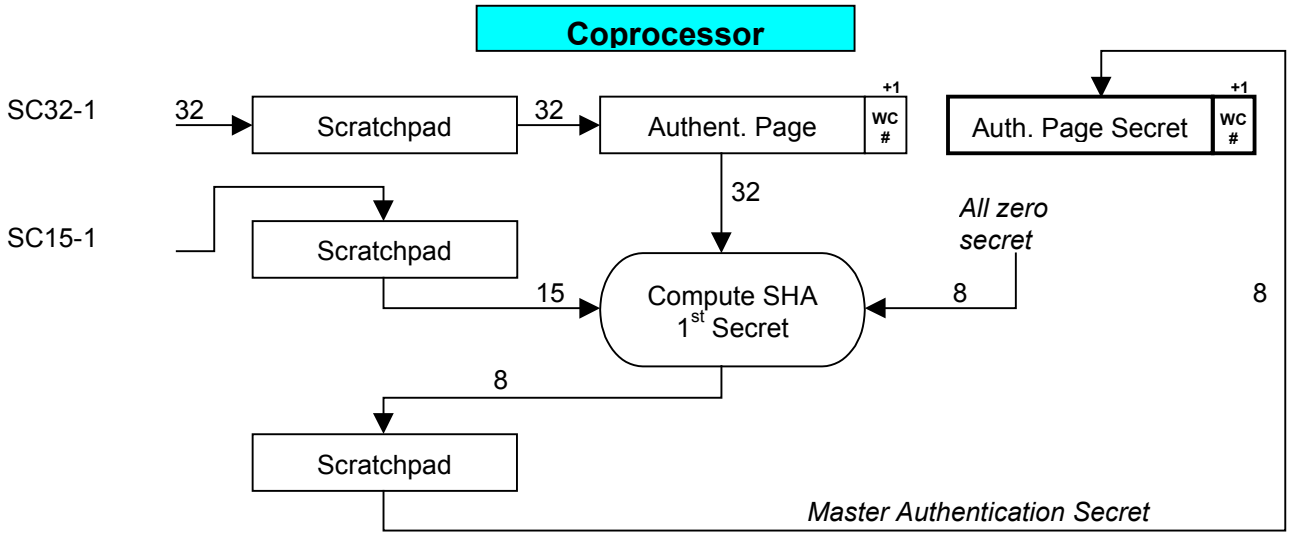
Step AI2

Title: Installation of initial Master Authentication Secret (MAS) in the coprocessor

Precondition:
- SC32-1 is defined
- SC15-1 is defined

Performed:
- When setting up the coprocessor for use in the system

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Using a dummy starting address, erase the scratchpad.	
2)	Using the starting address of the authentication page, write SC32-1 to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to authentication page.	
3)	Fill scratchpad locations 8 to 22 with SC15-1.	
4)	Using the starting address of the authentication page issue Compute SHA/1st Secret.	
5)	Using the starting address of the secret of the authentication page, write 8 dummy bytes to the scratchpad.	
6)	Using the starting address of the secret of the authentication page and a computed E/S byte, issue the Copy Scratchpad command.	
1)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
2)	To load the first 32 bytes of the Authentication Input Secret into the Authentication Page. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the remaining 15 bytes of the Authentication Input Secret into the scratchpad. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To compute a SHA-1 result from the content of the Authentication Page, 15 bytes of known scratchpad data and an all-zero secret. *The target address must point to a location within the Authentication Page.*
5)	To select the secret of the Authentication Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Authentication Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*

Ref. #	Purpose and *Comments*
6)	To make the SHA-1 result the secret of the Authentication Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

DS1963S Secret Address Assignments: (see step AI1)

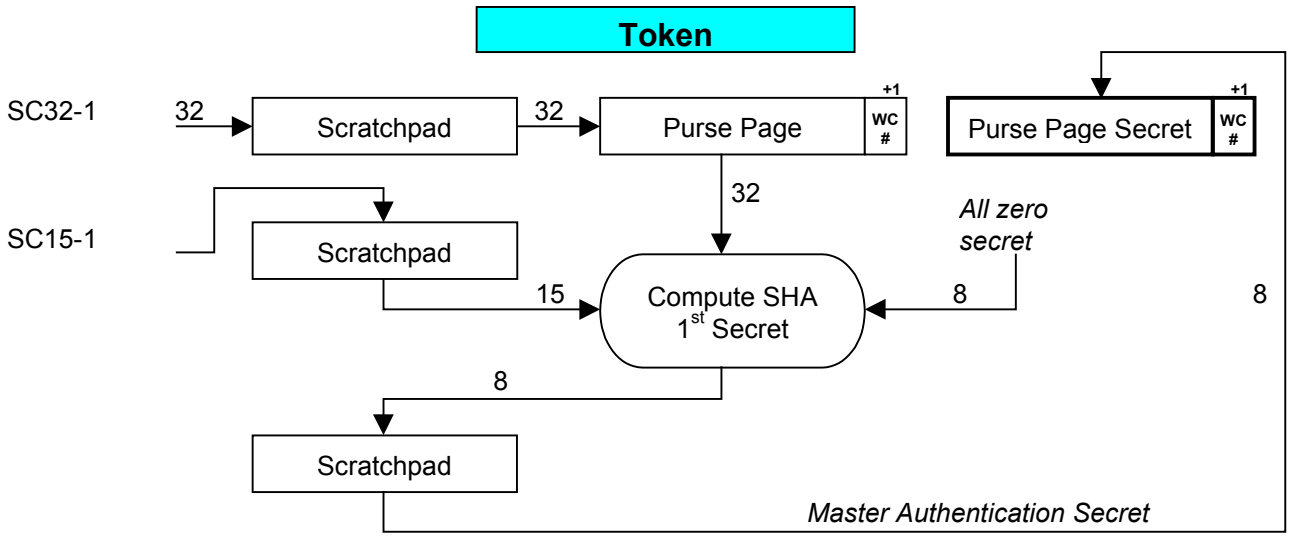
Step AI3

Title: Installation of initial Master Authentication Secret (MAS) with the purse page of the token

Precondition:
- SC32-1 is defined
- SC15-1 is defined
- The purse page number is defined

Performed:
- When setting up the token for use in the system

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Using a dummy starting address, erase the scratchpad.
2)		Using the starting address of the purse page, write SC32-1 to the scratchpad.
2)		Verify correct scratchpad writing and copy scratchpad data to purse page.
3)		Fill scratchpad locations 8 to 22 with SC15-1.
4)		Using the starting address of the purse page issue Compute SHA/1st Secret.

Ref. #	DS1963S Coprocessor	DS1963S Token
5)		Using the starting address of the secret of the purse page, write 8 dummy bytes to the scratchpad.
6)		Using the starting address of the secret of the purse page and a computed E/S byte, issue the Copy Scratchpad command.
1)		Using a dummy starting address, erase the scratchpad.

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
2)	To load the first 32 bytes of the Authentication Input Secret into the Purse Page. *Data is first written to the scratchpad, verified (e. g., read back) and then copied to the memory page. With unused tokens, the purse page number can be anywhere from page 9 to page 15. If a purse page already exists in the token, a new purse can be installed in an unused page. If the status of the token is unknown, first check the device directory to identify a vacant page before installing a new purse.*
3)	To load the remaining 15 bytes of the Authentication Input Secret into the scratchpad. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To compute a SHA-1 result from the content of the Purse Page, 15 bytes of known scratchpad data and an all-zero secret. *The target address must point to a location within the Purse Page.*
5)	To select the secret of the Purse Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Purse Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
6)	To make the SHA-1 result the secret of the Purse Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

DS1963S Purse Page Address Assignments:

Description	Address Range
Page 9	0120h to 013Fh
Page 10	0140h to 015Fh
Page 11	0160h to 017Fh
Page 12	0180h to 019Fh
Page 13	01A0h to 01BFh
Page 14	01C0h to 01DFh
Page 15	01E0h to 01FFh

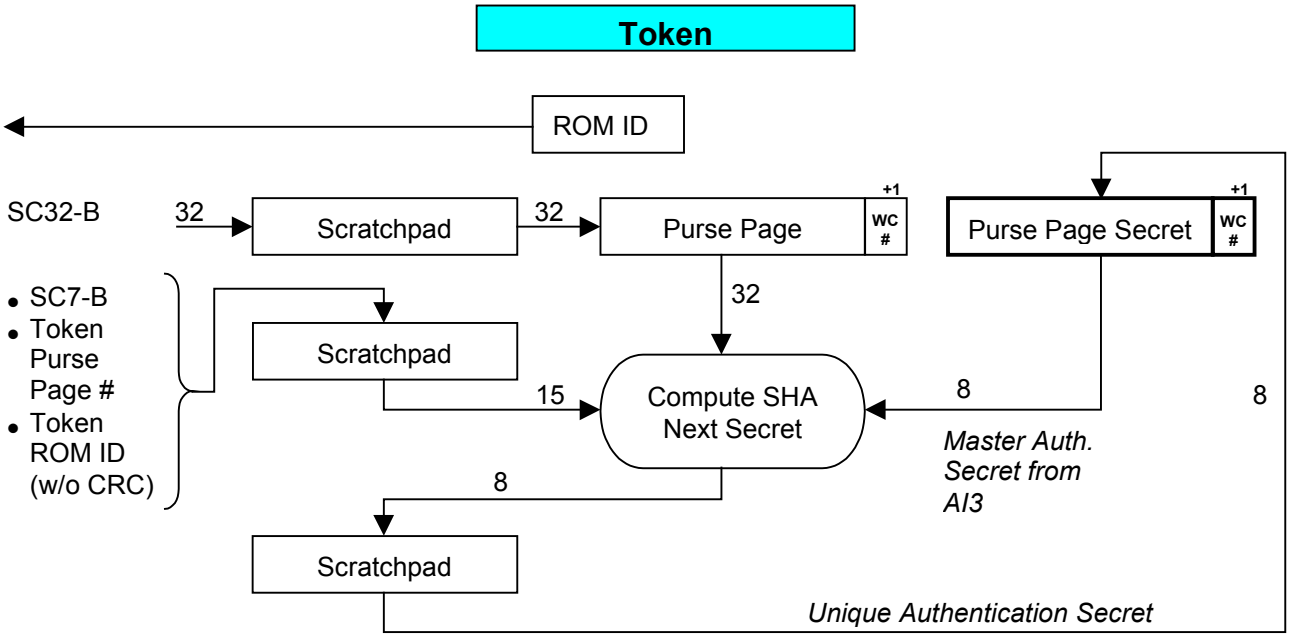
Step AI4

Title: Installation of the Unique Authentication Secret (UAS) with the purse page of the token. This step is also called "Binding the secret to the token".

Precondition:
- AI3 was performed successfully.
- SC32-B is defined
- SC7-B is defined
- The purse page number is defined

Performed:
- Before installing a purse file in the token

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Read ROM ID of token; verify correct reading
2)		Using the starting address of the purse page, write SC32-B to the scratchpad.
2)		Verify correct scratchpad writing and copy scratchpad data to purse page.
3)		Fill scratchpad locations 8 to 22 with first 4 bytes of SC7-B, purse page number, token ROM ID (without CRC), 3 remaining bytes of SC7-B
4)		Using the starting address of the purse page issue Compute SHA/Next Secret.
5)		Using the starting address of the secret of the purse page, write 8 dummy bytes to the scratchpad.

Ref. #	DS1963S Coprocessor	DS1963S Token
6)		Using the starting address of the secret of the purse page and a computed E/S byte, issue the Copy Scratchpad command.

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To create a secret that is unique to a particular device (token). *The ROM ID of the token is used as input to the SHA-1 computation.*
2)	To load the 32 bytes Binding Data into the Purse Page. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, and constant data. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b). Page number and ROM ID are also referred to as Binding Code.*
4)	To compute a SHA-1 result from the content of the Purse Page, 15 bytes of known scratchpad data and the current secret of the purse page. *The target address must point to a location within the Purse Page.*
5)	To select the secret of the Purse Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Purse Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
6)	To make the SHA-1 result the new secret of the Purse Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

DS1963S Purse Page Address Assignments: (see step AI3)

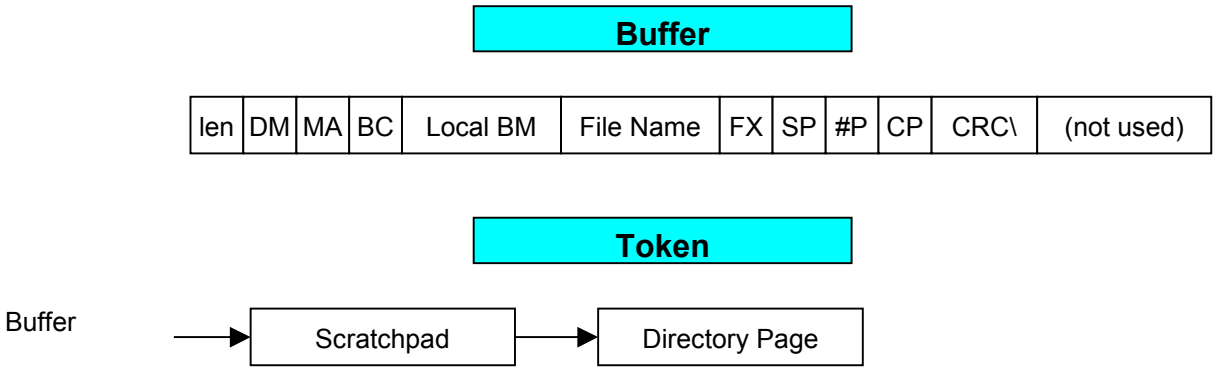
Step AI5

Title: Installation of a device file directory with an entry for the purse file in the token.

Precondition:
- The file name of the purse file to be created is defined
- The page number (location) of the purse file to be created (length = 1 page) is defined
- A device directory has not yet been created in the token.

Performed:
- When initializing (commissioning) a token for use in the application.

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Using a dummy starting address, erase the scratchpad.
A 2)		Using the starting address of page 0, write to scratchpad the initialization string of a device directory (see AN114) with the entry of the purse file (name, page #, length).
2)		Verify correct scratchpad writing and copy scratchpad to memory.
3)		Read the directory page data from the token.
3)		Compare the directory page data read from the token to the expected data. If the data doesn't match, go to A.

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. *The target address issued is not relevant; any value is accepted.*
2)	To install a file directory in the Directory Page (page 0). *See Table 1 for the directory format. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page. This description assumes that the token is unused, i. e., no service record exists. Additional service records or purses may be installed later. The device supports a total of 7 service records. When installing an additional purse or service record, the data for the directory page is created by first reading the existing (potentially multi-page) directory and adding another file entry to it. This always affects the last page of the device directory and may extend the directory by one partially filled page. See AN114 for the device directory format and definitions. Every additional service record requires the installation of a service-specific Master Authentication Secret and Unique Authentication Secret with the new record in the token (steps AI3, AI4).*
3)	To verify whether the directory was installed properly. *If necessary, the directory installation is repeated.*

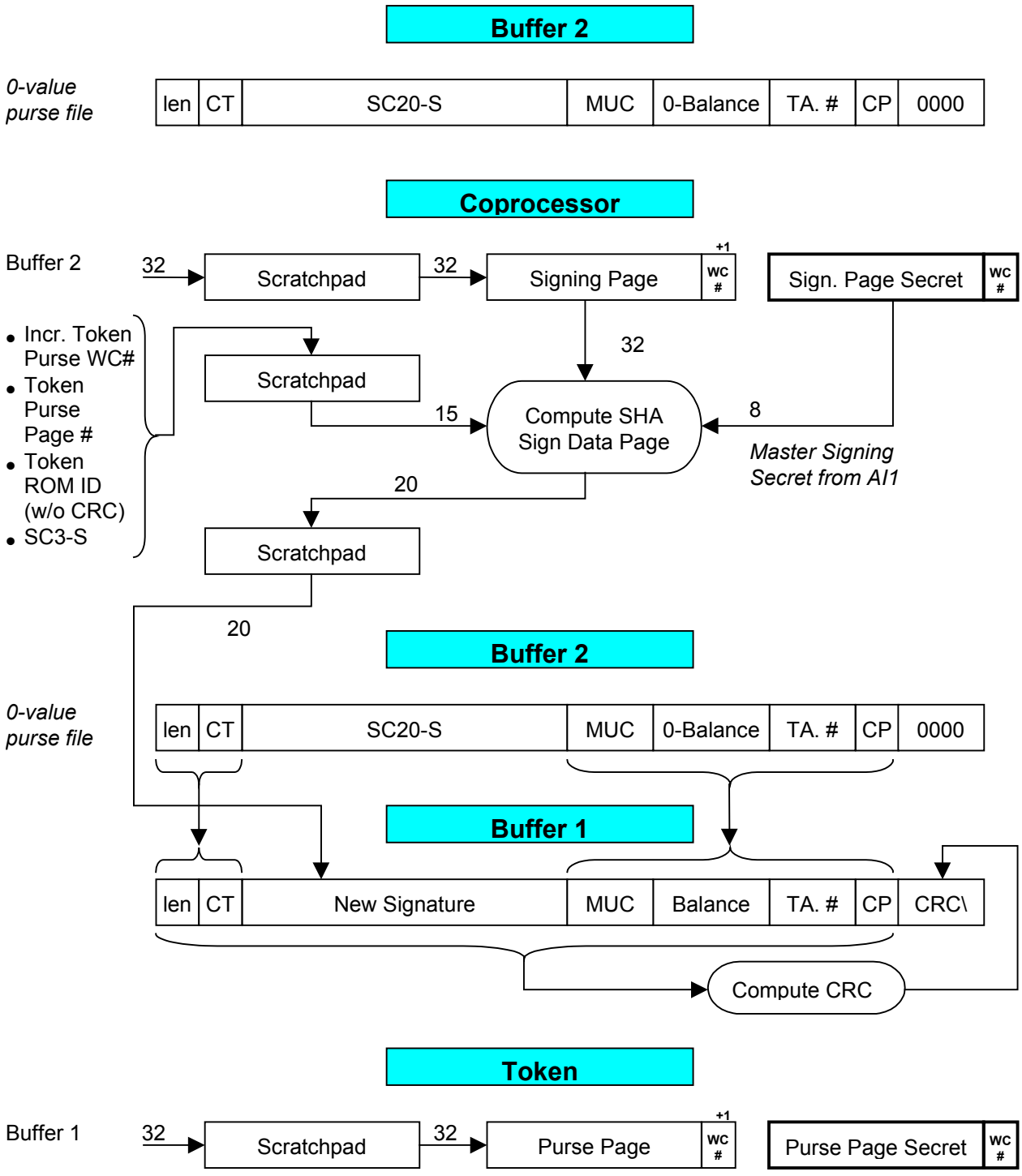
Step AI6

Title: Writing a zero-value purse file to the token

Precondition:
- AI4 was performed successfully.
- AI5 was performed successfully.
- Name and location of the purse file are known (from AI5).
- The format and contents of the zero-value purse file are known.

Performed:
- Immediately before releasing the token for use in the application.

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Perform step AF1 to install the purse page's Unique Authentication Secret as the secret of the workspace page of the coprocessor.	
2)	Perform step AF2 to check whether the purse page's secret is valid in the system.	
3)	Create the data of a zero-value purse file using SC20-S as embedded signature and 00-bytes as page CRC	
4)	Using the starting address of the signing page, write the zero-value purse file to the scratchpad.	
4)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
5)	Write to scratchpad locations 8 to 22: (data page address don't care) incremented token purse page write cycle count, token purse page number, token ROM ID (without CRC), SC3-S.	
6)	Using the starting address of the signing page issue Compute SHA/Sign Data Page. Now the scratchpad locations 8 to 27 contain the SHA-1 result which serves as a signature of the new purse.	
7)	Take the data as written to the signing page and replace SC20-S with the computed signature from the scratchpad of the coprocessor.	
7)	Replace the all-zero CRC with a computed CRC according to the rules specified in AN114. The resulting data is the updated and formatted purse file for the token.	
8)		Using the starting address of the purse file in the token, write the new purse file to the scratchpad of the token.
8)		Verify correct scratchpad writing and copy scratchpad data to purse page.
9)	Perform step AF5 to verify successful installation of the zero-value purse file in the token that it was computed for.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare for the verification of the purse page's unique secret. *If the token does not belong to the system, the verification will fail.*
2)	To verify whether the secret with the purse page in the token follows the rules that were defined for the application. *It is assumed that the purse's secret (UAS) has been installed and that the purse file has an entry in the device directory.*
3)	To prepare the creation of a valid (though empty) purse file for the particular token. *See Table 2 for the purse file format. For simplicity, the CRC16 at the end of the purse is replaced by zeros.*
4)	To load the zero-value purse file with embedded initial signature into the Signing Page (page 8). *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
5)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the incremented write-cycle counter of the purse page, and the Signing Challenge. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b). The token ROM ID and purse page write cycle count were obtained in steps AF1, AF2.*
6)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor.*

Ref. #	Purpose and *Comments*
7)	To create the purse file for the token with embedded personalized signature and valid CRC16. *The purse file has to meet the formal requirements of the 1-Wire File Structure.*
8)	To write the purse file to the purse page in the token. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
9)	To verify whether token was not swapped before the purse file was written to the token. *If the token was swapped, the verification will fail. The signature in the purse file is only valid in the particular token and memory page with the write cycle count it was computed for.*

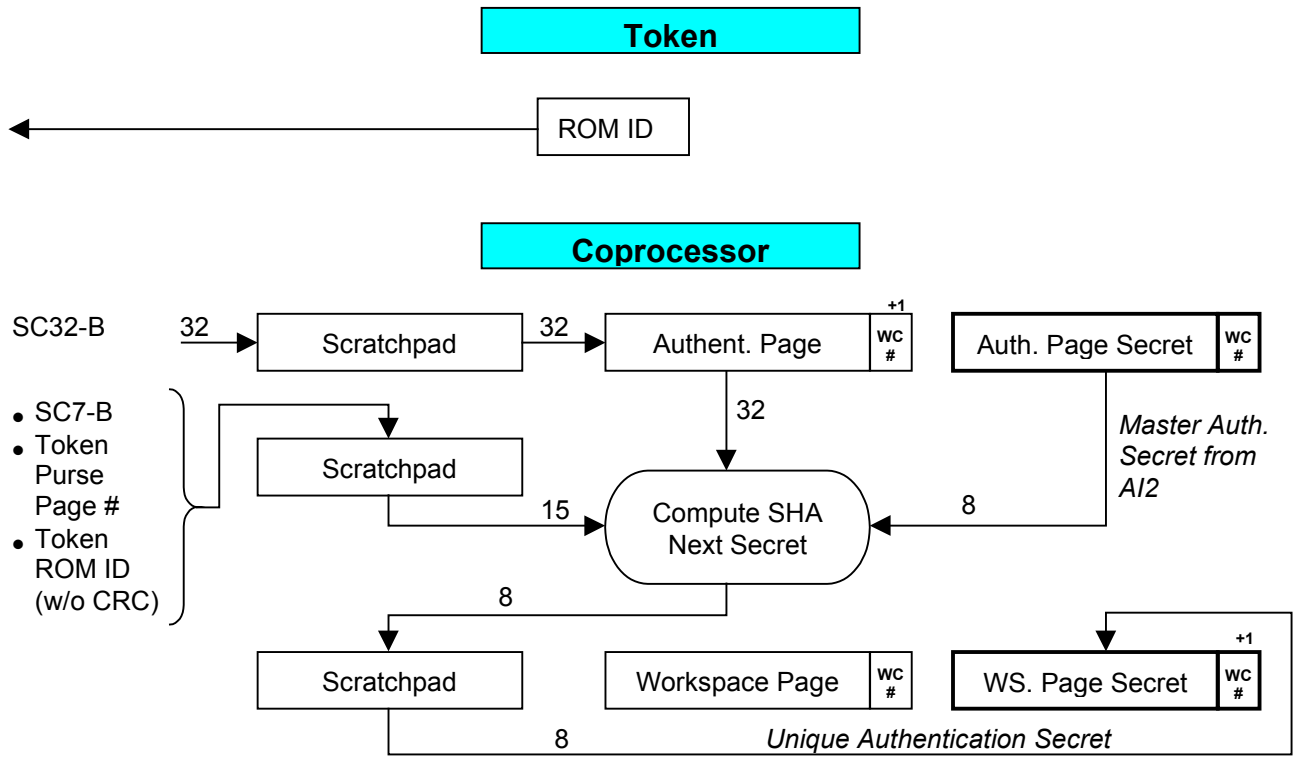
Step AF1

Title: Installation of the purse's UAS as the secret of the workspace page of the coprocessor

Precondition:
- AI2 was performed successfully on the coprocessor
- AI4 was performed successfully
- The purse page number to be used in this and the following steps is known.

Performed:
- Preparation for token purse page authentication

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Read ROM ID of token; verify correct reading.
2)		Using a dummy starting address, erase the scratchpad.
2)	Using a dummy starting address, erase the scratchpad.	
3)	Using the starting address of the authentication page, write SC32-B to the scratchpad.	
3)	Verify correct scratchpad writing and copy scratchpad data to authentication page.	
4)	Fill scratchpad locations 8 to 22 with first 4 bytes of SC7-B, token purse page number, token ROM ID (without CRC), 3 remaining bytes of SC7-B	
5)	Using the starting address of the authentication page issue Compute SHA/Next Secret.	
6)	Using the starting address of the secret of the workspace page, write 8 dummy bytes to the scratchpad.	
7)	Using the starting address of the secret of the workspace page and a computed E/S byte, issue the Copy Scratchpad command.	
2)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare computing the unique secret of the purse page. *The ROM ID of the token is used as input to the SHA-1 computation.*
2)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
3)	To load the 32 bytes Binding Data into the Authentication Page. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
4)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, and constant data. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
5)	To compute the unique authentication secret (UAS) of the particular purse. *The target address must point to a location within the Authentication Page.*
6)	To select the secret of the Workspace Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Workspace Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
7)	To install the UAS as the secret of the Workspace Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

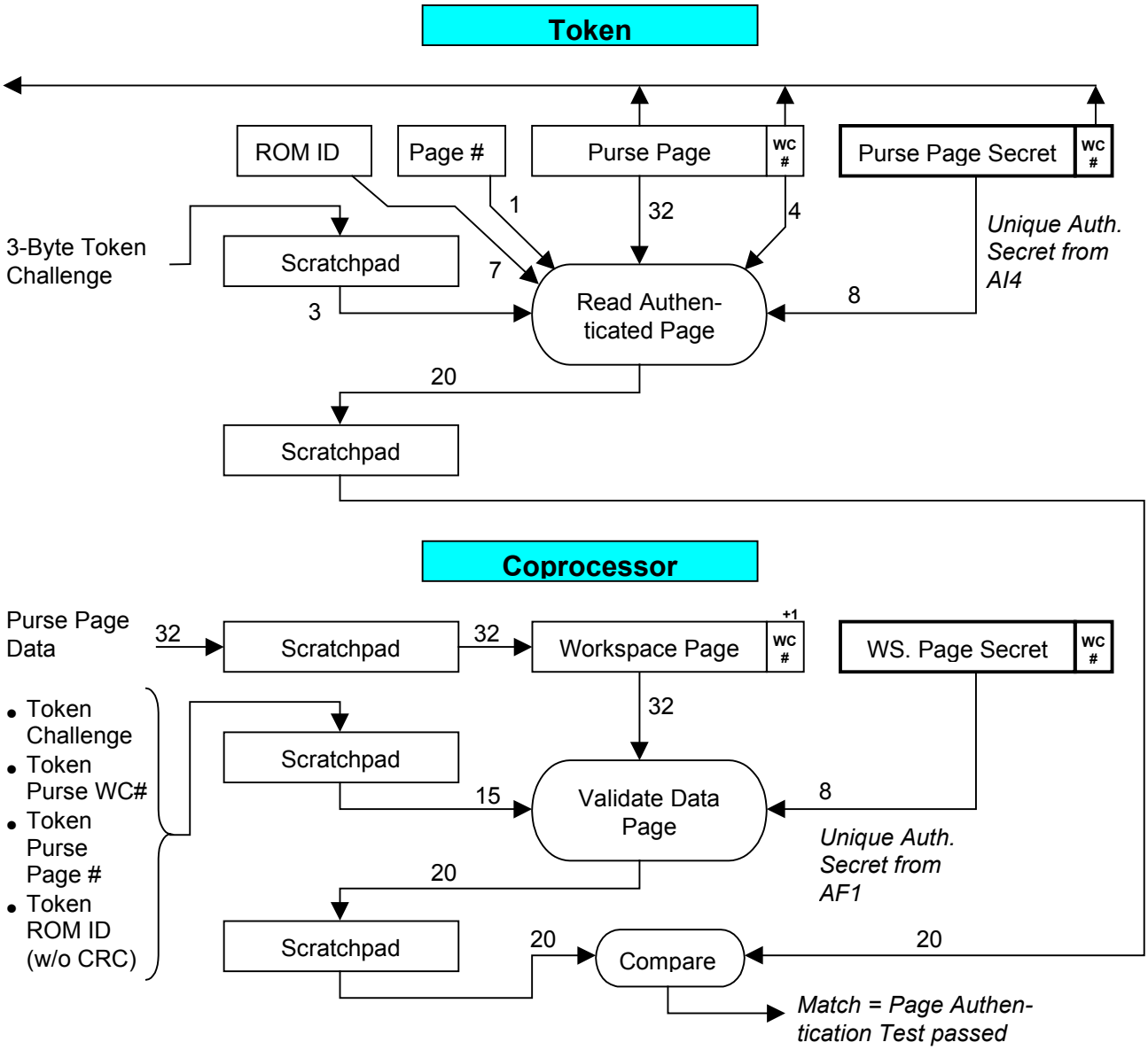
Step AF2

Title: Verify whether the purse page's secret is valid in the system

Precondition:
- AF1 was performed successfully
- AI6 or AF4 was performed successfully
- The purse page number to be used in this and the following steps is known.

Performed:
- Before checking the purse data

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Write a 3-byte random number to scratchpad locations 20 to 22. (The page address used is not relevant.) This number will be used as challenge with the next command.
1)		Perform a Read Authenticated Page Command using the starting address of the page that contains the purse file. Error-check and save the page data and page write cycle counter value.
1)		Read the SHA-1 result from the scratchpad and save it in a buffer.
2)	Using a dummy starting address, erase the scratchpad.	
3)	Using the starting address of the workspace page, write the page data read from the token to the scratchpad.	
3)	Verify correct scratchpad writing and copy scratchpad data to workspace page.	
4)	Write to scratchpad locations 8 to 22: (data page address don't care) token purse page write cycle count, token purse page number, token ROM ID (without CRC), the same random number that was used with Read Authenticated Page.	
5)	Using the starting address of the workspace page issue the Validate Data Page command.	
5)	Take the SHA-1 result from the Read Authenticated Page command and use it with the Match Scratchpad command. If this command results in AAh pattern, the SHA results did match, confirming that the purse pages secret is valid in the system.	
2)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To read the purse page including write cycle counter. To obtain a SHA-1 result from the token based on a 3-byte challenge, purse data, token ROM ID, Purse Page number, purse page write cycle count, and UAS of the purse. *Using a random challenge generates different SHA-1 results from otherwise identical input data. Only the legitimate token can perform the correct SHA-1 computation.*
2)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
3)	To load the 32 bytes Purse Data into the Workspace Page. *This is a precondition for re-computing the SHA-1 result that was read from the token. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
4)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the purse write-cycle counter, and the challenge that was used when reading the purse. *This is a precondition for re-computing the SHA-1 result that was read from the token. The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
5)	To re-compute the SHA-1 result that was read from the token. To compare the SHA-1 result from the token to the one computed in the coprocessor. *The target address must point to a location within the Workspace Page. If both SHA-1 results match, the secret of the purse was computed according to the rules that were defined for the application. This is the evidence that the purse belongs to the system.*

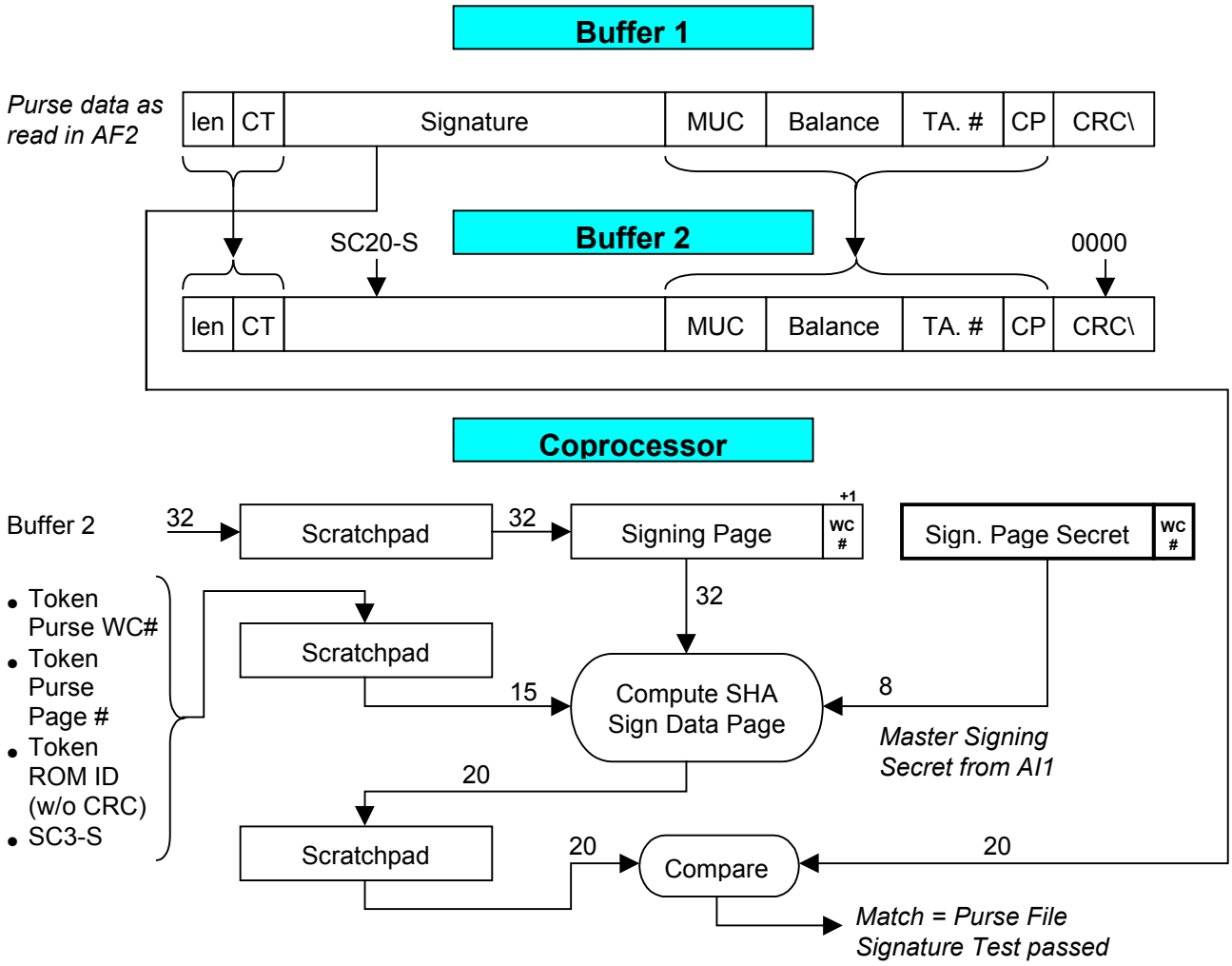
Step AF3

Title: Verify whether the signature embedded in the purse is valid

Precondition:
- AF2 was performed successfully immediately prior to this step.

Performed:
- Before updating the purse

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Take the purse file read in the previous step, replace the embedded signature with SC20-S, and replace the CRC16 at the end of the purse file with all zeros.	
2)	Using the starting address of the signing page, write the modified purse file to the scratchpad.	

Ref. #	DS1963S Coprocessor	DS1963S Token
2)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
3)	Write to scratchpad locations 8 to 22: (data page address don't care) token purse page write cycle count, token purse page number, token ROM ID (without CRC), SC3-S.	
4)	Using the starting address of the signing page issue the Sign Data Page command.	
5)	Use the Match scratchpad command to compare the SHA result found in the scratchpad of the coprocessor (locations 8 to 22) to the one that was embedded in the purse file read from the token. If the SHA-1 results match, the purse data is authentic.	
6)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare checking the validity of the signature embedded in the purse file of the particular token. *The signature needs to be replaced by the initial signature for verifying the embedded signature.*
2)	To load the purse file with embedded initial signature into the Signing Page (page 8). *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the current write-cycle counter of the purse page, and the Signing Challenge. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor. It should match the embedded signature of the purse file.*
5)	To check whether the embedded signature and the newly computed signature are identical; a match confirms that the purse file is valid and belongs to the particular token. *Using the Match Scratchpad command is more efficient than reading the SHA-1 result and then comparing it in the host. A valid signature is assumed to imply a valid data structure in the purse file.*
6)	To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*

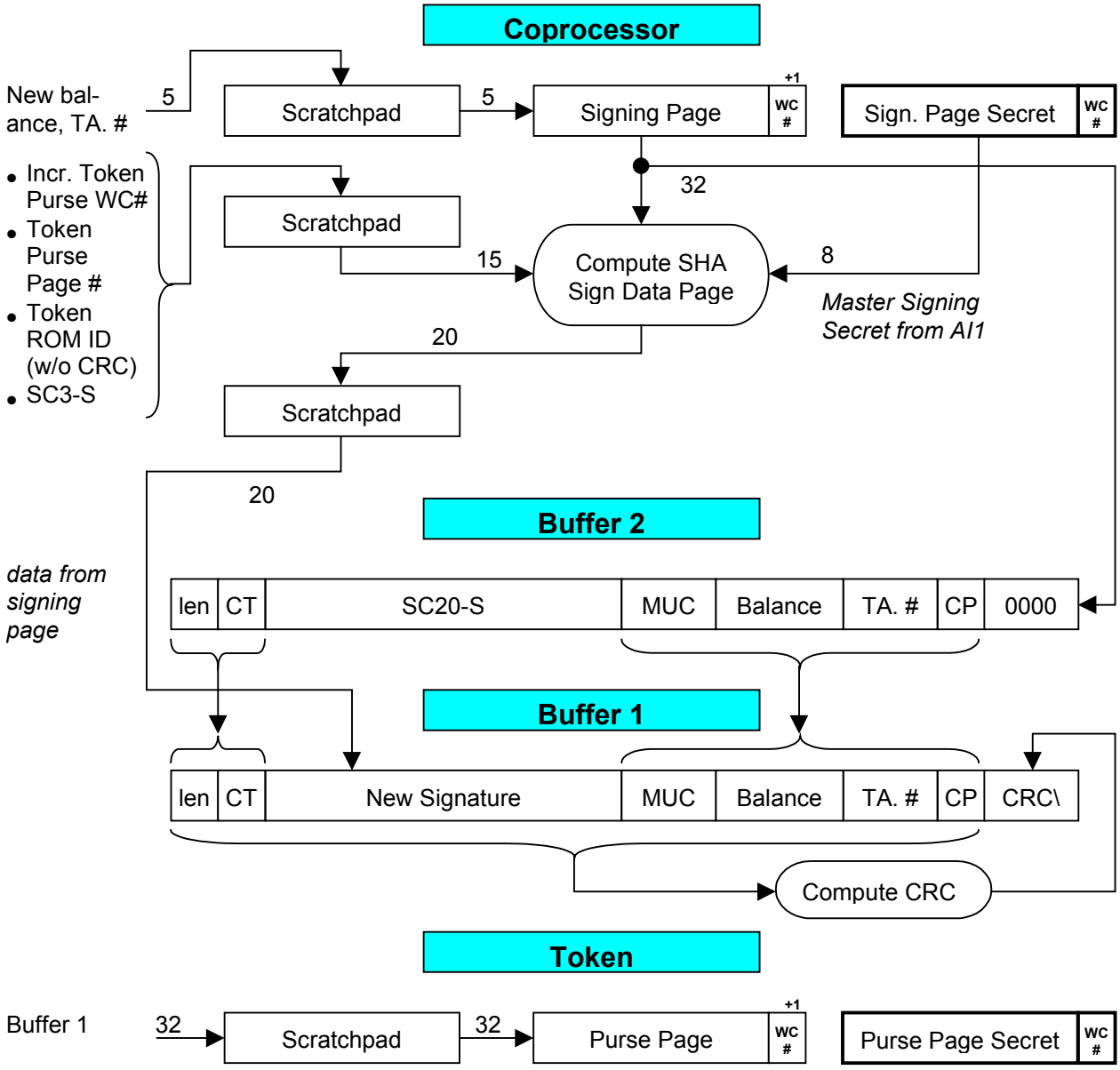
Step AF4

Title: Updating the purse file in the token

Precondition:
- AF3 was performed successfully prior to this step
- The new purse value has been determined.
- The signing page of the coprocessor contains the purse data as of step AF3
- There was no write access to the purse page in the token since the purse was last read.

Performed:
- Precondition for releasing the purchased goods

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Using the appropriate starting address within the signing page, write to the scratchpad the sections of the purse data that will change (i. e., balance, transaction #).	
1)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
2)	Write to scratchpad locations 8 to 22: (data page address don't care) incremented token purse page write cycle count, token purse page number, token ROM ID (without CRC), SC3-S.	

Ref. #	DS1963S Coprocessor	DS1963S Token
3)	Using the starting address of the signing page issue Compute SHA/Sign Data Page. Now the scratchpad locations 8 to 27 contain the SHA-1 result which serves as a signature of the updated purse.	
4)	Read the data from the signing page and replace the embedded SC20-S with the computed signature from the scratchpad of the coprocessor.	
4)	Replace the all-zero CRC with a computed CRC according to the rules specified in AN114. The resulting data is the updated and formatted purse file for the token.	
5)		Using the starting address of the purse file in the token, write the updated purse file to the scratchpad of the token.
5)		Verify correct scratchpad writing and copy scratchpad data to purse page.
6)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To update the purse with the after-purchase value and a new transaction ID. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
2)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the incremented write-cycle counter of the purse page, and the Signing Challenge. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
3)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor.*
4)	To create the purse file for the token with embedded personalized signature and valid CRC16. *The purse file has to meet the formal requirements of the 1-Wire File Structure.*
5)	To write the purse file to the purse page in the token. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
6)	To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*

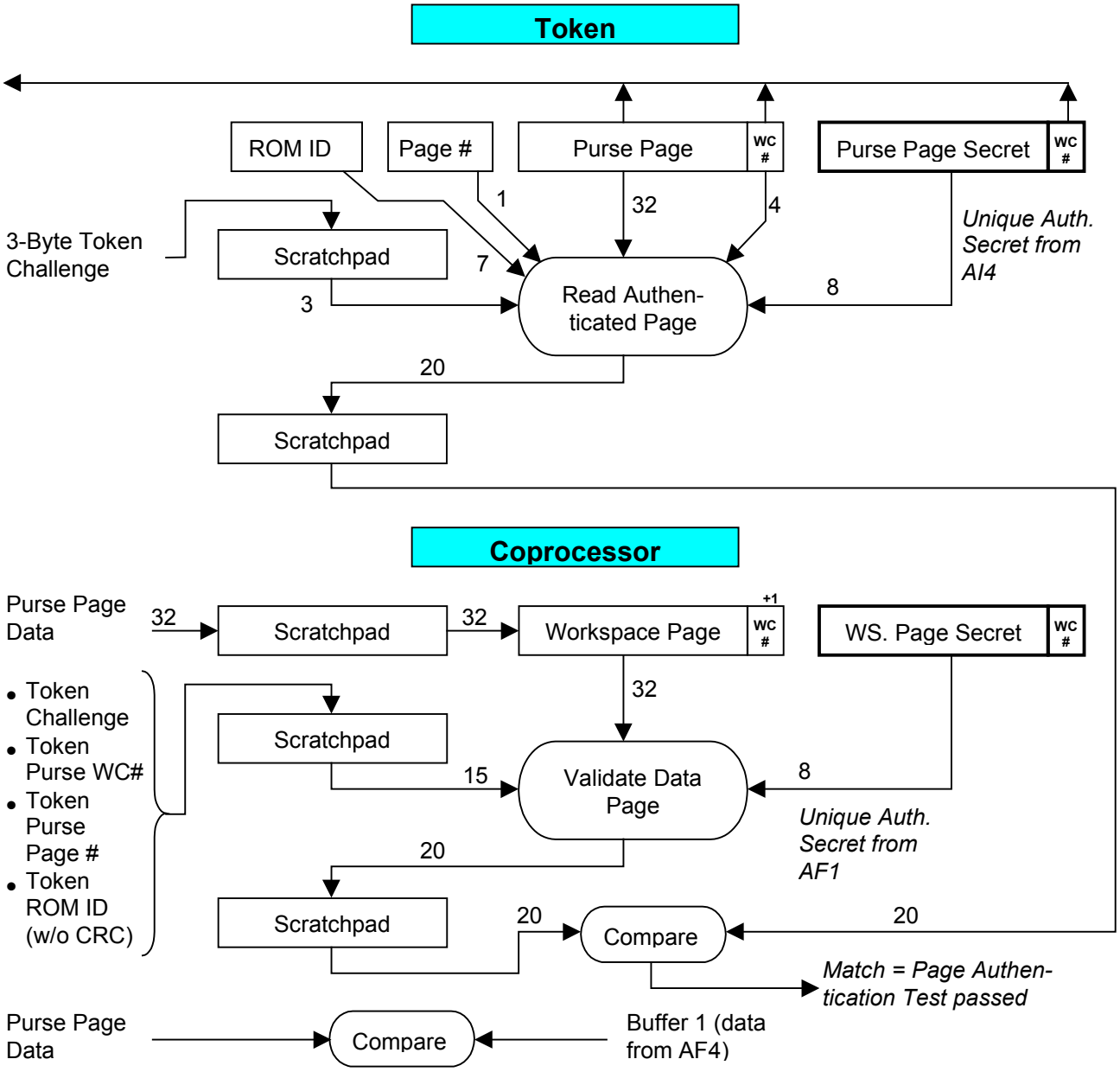
Step AF5

Title: Verify whether the updated purse file was written successfully to the same token that was read before.

Precondition:
- AF4 was performed successfully
- The expected content of the purse file is known.

Performed:
- Immediately before dispensing goods

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)		Write a 3-byte random number to scratchpad locations 20 to 22. (The page address used is not relevant.) This number will be used as challenge with the next command.
1)		Perform a Read Authenticated Page Command using the starting address of the page that contains the purse file. Error-check and save the page data and page write cycle counter value.
1)		Read the SHA-1 result from the scratchpad and save it in a buffer.

Ref. #	DS1963S Coprocessor	DS1963S Token
2)	Using the starting address of the workspace page, write the page data read from the token to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to workspace page.	
3)	Write to scratchpad locations 8 to 22: (data page address don't care) token purse page write cycle count, token purse page number, token ROM ID (without CRC), the same random number that was used with Read Authenticated Page.	
4)	Using the starting address of the workspace page issue the Validate Data Page command.	
4)	Take the SHA-1 result from the Read Authenticated Page command and use it with the Match Scratchpad command. If this command results in AAh pattern, the SHA results did match, confirming that the purse page's secret is valid in the system.	
5)	Compare the data read from the token to the expected purse file data. If it is the same token as before, but the data doesn't match, repeat the token writing steps of AF4 (or AI6, respectively).	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To read the purse page including write cycle counter. To obtain a SHA-1 result from the token based on a 3-byte challenge, purse data, token ROM ID, Purse Page number, purse page write cycle count, and UAS of the purse. *Using a random challenge generates different SHA-1 results from otherwise identical input data. Only the legitimate token can perform the correct SHA-1 computation.*
2)	To load the 32 bytes Purse Data into the Workspace Page. *This is a precondition for re-computing the SHA-1 result that was read from the token. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the purse write-cycle counter, and the challenge that was used when reading the purse. *This is a precondition for re-computing the SHA-1 result that was read from the token. The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To re-compute the SHA-1 result that was read from the token. To compare the SHA-1 result from the token to the one computed in the coprocessor. *The target address must point to a location within the Workspace Page. If both SHA-1 results match, the secret of the purse was computed according to the rules that were defined for the application. This is the evidence that the purse belongs to the system.*
5)	To verify whether the updated purse file (after-purchase value) was actually written to the token. *From the previous step (AF4) the host still knows the expected new purse data. Only if the purse was updated will the merchandise be released.*

APPENDIX B

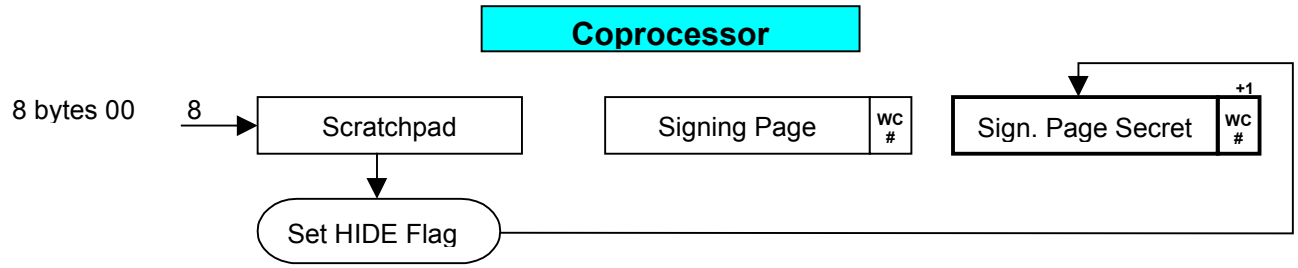
Step BI1

Title: Installation of an all-zero secret for the signing page of the coprocessor

Precondition:
- none

Performed:
- Before installing the initial Master Authentication Secret in a token

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)	Using a dummy starting address, erase the scratchpad.	
2)	Using the starting address of any data memory page (page number 0 to 15) write eight 00-bytes to the scratchpad.	
2)	Verify correct scratchpad writing.	
3)	Remove the coprocessor from its socket and reinsert it after a few seconds.	
4)	Using the starting address of the signing page secret, write eight dummy bytes to the scratchpad.	
5)	Using the starting address of the signing page secret and a computed E/S byte, issue the Copy Scratchpad command.	
1)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
2)	To prepare the installation of a loaded (not computed) secret in the coprocessor. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page. The compute 1st secret function of the coprocessor uses an all-zero secret to start with; the DS1961S token can load anything as the first secret. For simplicity, the initialization of the token is chosen to comply with the DS1963S scheme. As a consequence, the coprocessor needs to have an all-zero secret for the signing page to compute the SHA-1 result that is required by the token to write the authentication input secret to the binding page.*
3)	To set the HIDE flag. *The HIDE flag must be set to copy data from the scratchpad to a secret. If the DS1963S is located behind a DS2409 MicroLAN Coupler, it is not necessary to remove the device from the socket to set the HIDE flag. Instead, use the DS2409's Discharge Lines command. This generates a Power-On-Reset for the DS1963S, which sets the HIDE flag, using only standard 1-Wire commands.*
4)	To select the secret of the Signing Page as the destination of the scratchpad data. *The target address must point to a location within the secret of the Signing Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
5)	To copy the scratchpad data to the secret of the Signing Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

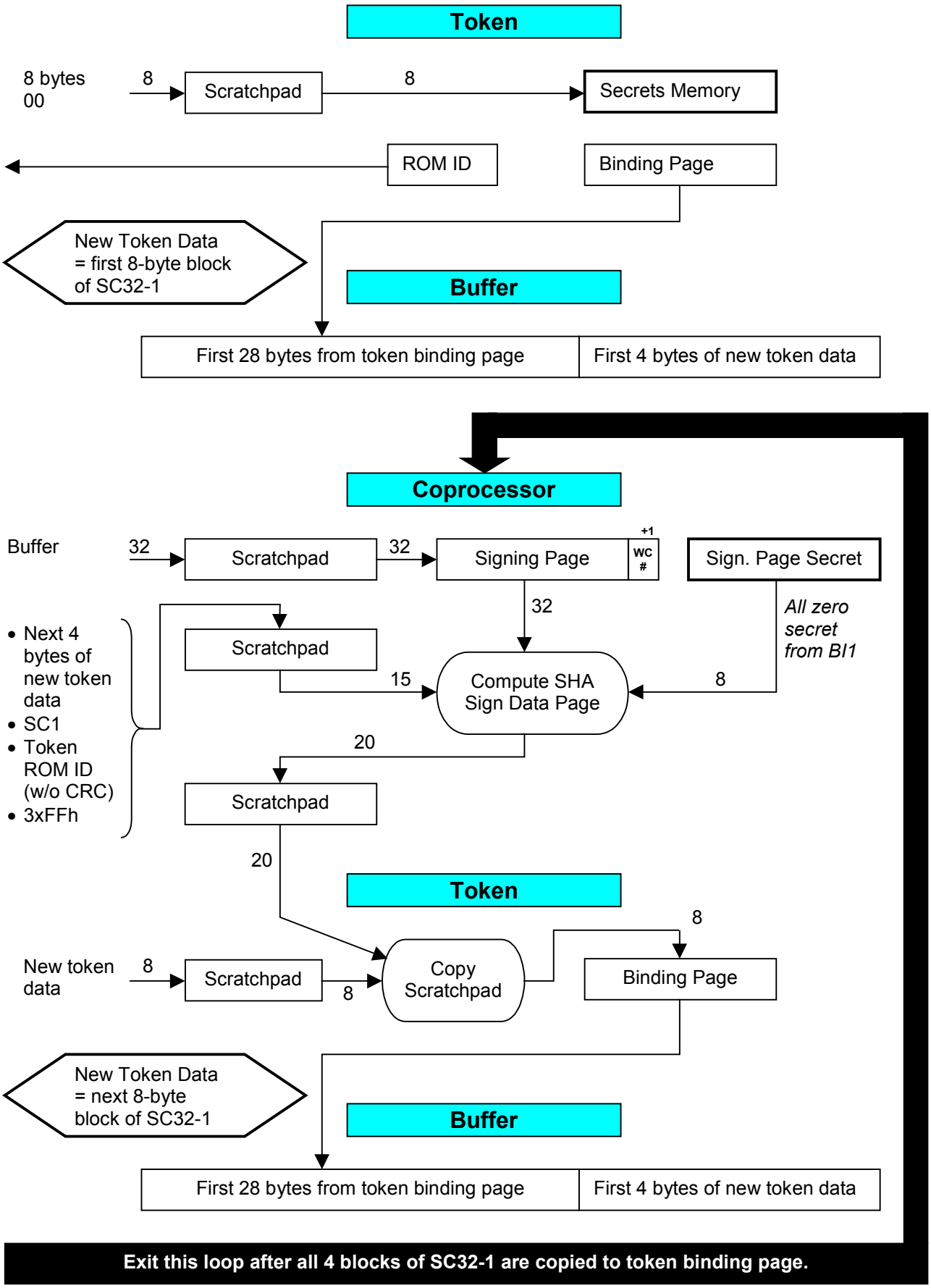
Step BI2

Title: Installation of the initial Master Authentication Secret (MAS) in the token. This requires writing SC32-1 to the token binding page.

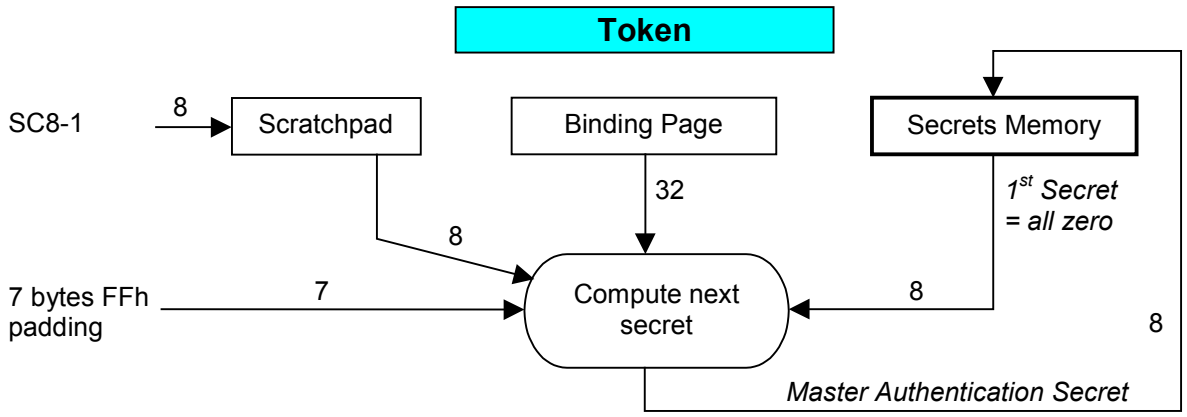
Precondition:
- BI1 was performed successfully on the coprocessor used for this step.
- SC32-1 is defined
- SC8-1 is defined
- The token is not write-protected.

Performed:
- Before installing Unique Authentication Secret (UAS) in the token

Data Flow Diagram:



Data Flow Diagram (continued):



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Write eight 00 bytes to the token's scratchpad using the starting address of the secret and verify correct scratchpad writing.
1)		Issue the Load First Secret command with the correct target address and a computed E/S byte.
2)		Read ROM ID of token; verify correct reading
2)		Read data of token binding page; verify correct reading.
3)	Using a dummy starting address, erase the scratchpad.	
4)	For the new data to be written to the token use the first 8 bytes of SC32-1.	
A 5)	Using the starting address of the signing page, write the first 28 bytes of the token binding page to the scratchpad. For the remaining four bytes of the scratchpad use the first four bytes of the new token data.	
5)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
6)	Write to scratchpad locations 8 to 22: (data page address don't care) the next four bytes of the new token data, SC1 (token binding page number), token ROM ID (without CRC), 3 bytes FFh.	
7)	Using the starting address of the signing page issue Compute SHA/Sign Data Page. Now the scratchpad locations 8 to 27 contain the SHA-1 result needed to write the particular 8 bytes to the token.	
8)		Write the new token data to the token's scratchpad using the correct target starting address and verify correct scratchpad writing
8)		Issue the copy scratchpad command with the correct target address and a computed E/S byte and send the SHA-1 result computed by the co-processor.

Ref. #	DS1963S Coprocessor	DS1961S Token
9)		Read the full content of the binding page and store it in a buffer. Go to A and continue using the next 8-byte chunk of SC32-1 until the whole SC32-1 is written to the token binding page.
10)		Write SC8-1 to the scratchpad using the starting address of the token binding page.
11)		Issue the Compute Next Secret command with the correct starting address.

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To install the initial all-zero secret in the token. (See Note 2) *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the secrets memory.*
2)	To prepare the computation of a SHA-1 result that is required by the token to copy scratchpad data to memory. *The ROM ID of the token and the data of the token's binding page are used as input for the SHA-1 computation.*
3)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
4)	To split the new data for the memory page into 8-byte chunks that fit into the scratchpad. *The size of the DS1961S scratchpad is eight bytes.*
5)	To prepare the computation of a SHA-1 result that is required by the token to copy scratchpad data to memory. *The signing page needs to hold the first 28 bytes of the token's memory page that is to be written to plus the first four new bytes for that memory page. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
6)	To load the scratchpad with the remaining 15 bytes that are needed by the coprocessor to compute a SHA-1 result that is required by the token to copy scratchpad data to memory. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
7)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor. It is required by the token to copy scratchpad data to memory.*
8)	To write one 8-byte segment of the new page data to the binding page in the token. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page. Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command. The execution of this command requires the SHA-1 result of Note 7).*
9)	To prepare the computation of the next SHA-1 result that is required by the token to copy scratchpad data to memory. *The memory data of the token's binding page is used as input to the SHA-1 computation. Since the page data has changed, the new data needs to be loaded for the SHA-1 computation (Note 5). Reading the data page allows the verification of successful execution of the copy scratchpad command.*
10)	To load the SC8-1 subset of the Authentication Input Secret into the scratchpad. *The Compute Next Secret command extends SC8-1 with FFh bytes to form SC15-1. For this reason, seven bytes of SC15-1 (the portion of the authentication input secret that is loaded into the scratchpad of the coprocessor) need to be FFh.*
11)	To compute the initial master authentication secret from the content of the binding page, known scratchpad data and an all-zero secret, and install the new secret in the token. *The target address must point to a location within the Binding Page. The Authentication Input Secret is erased from the binding page when the Unique Authentication Secret (UAS) is installed in step BI4.*

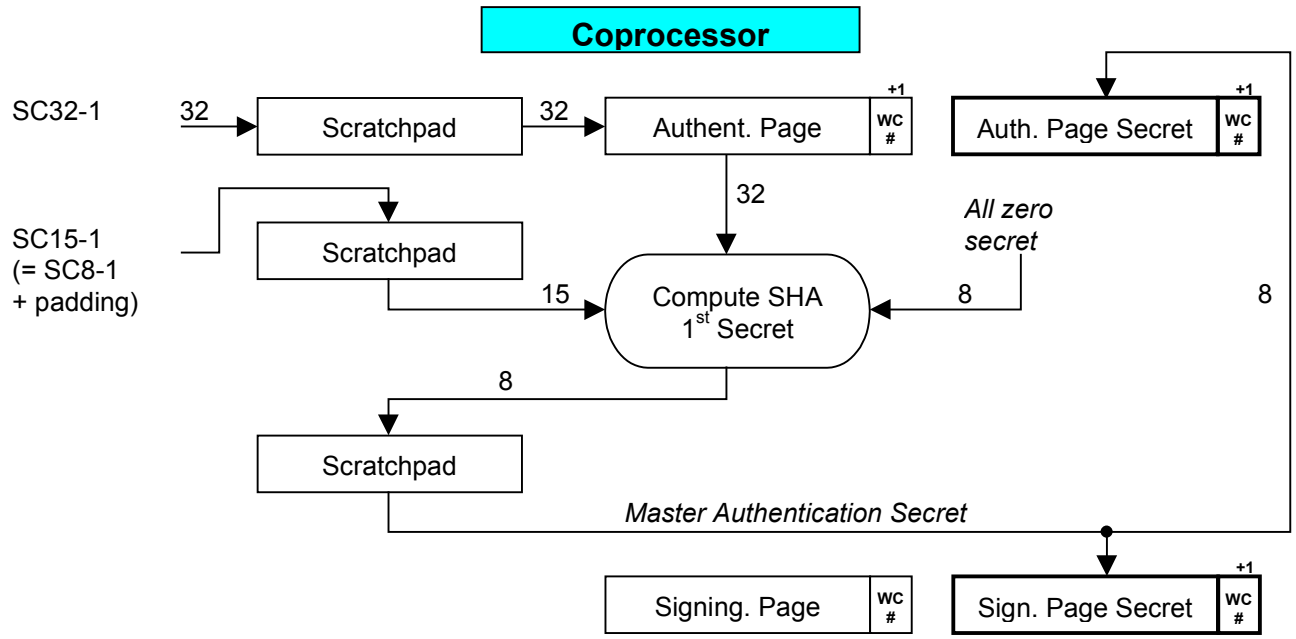
Step BI3

Title: Installation of Initial Master Authentication Secret (MAS) in the coprocessor

Precondition:
- SC32-1 is defined
- SC15-1 is defined

Performed:
- When setting up the coprocessor for use in the system

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1963S Token
1)	Using a dummy starting address, erase the scratchpad.	
2)	Using the starting address of the authentication page, write SC32-1 to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to authentication page.	
3)	Fill scratchpad locations 8 to 22 with SC15-1.	
4)	Using the starting address of the authentication page issue Compute SHA/1st Secret.	
5)	Using the starting address of the secret of the authentication page, write 8 dummy bytes to the scratchpad.	
6)	Using the starting address of the secret of the authentication page and a computed E/S byte, issue the Copy Scratchpad command.	

Ref. #	DS1963S Coprocessor	DS1963S Token
7)	Using the starting address of the secret of the signing page, write 8 dummy bytes to the scratchpad.	
8)	Using the starting address of the secret of the signing page and a computed E/S byte, issue the Copy Scratchpad command.	
1)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
2)	To load the first 32 bytes of the Authentication Input Secret into the Authentication Page. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the remaining 15 bytes of the Authentication Input Secret into the scratchpad. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To compute a SHA-1 result from the content of the Authentication Page, 15 bytes of known scratchpad data and an all-zero secret. *The target address must point to a location within the Authentication Page.*
5)	To select the secret of the Authentication Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Authentication Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
6)	To make the SHA-1 result the secret of the Authentication Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*
7)	To select the secret of the Signing Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the* Signing *Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
8)	To make the SHA-1 result the secret of the Signing Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

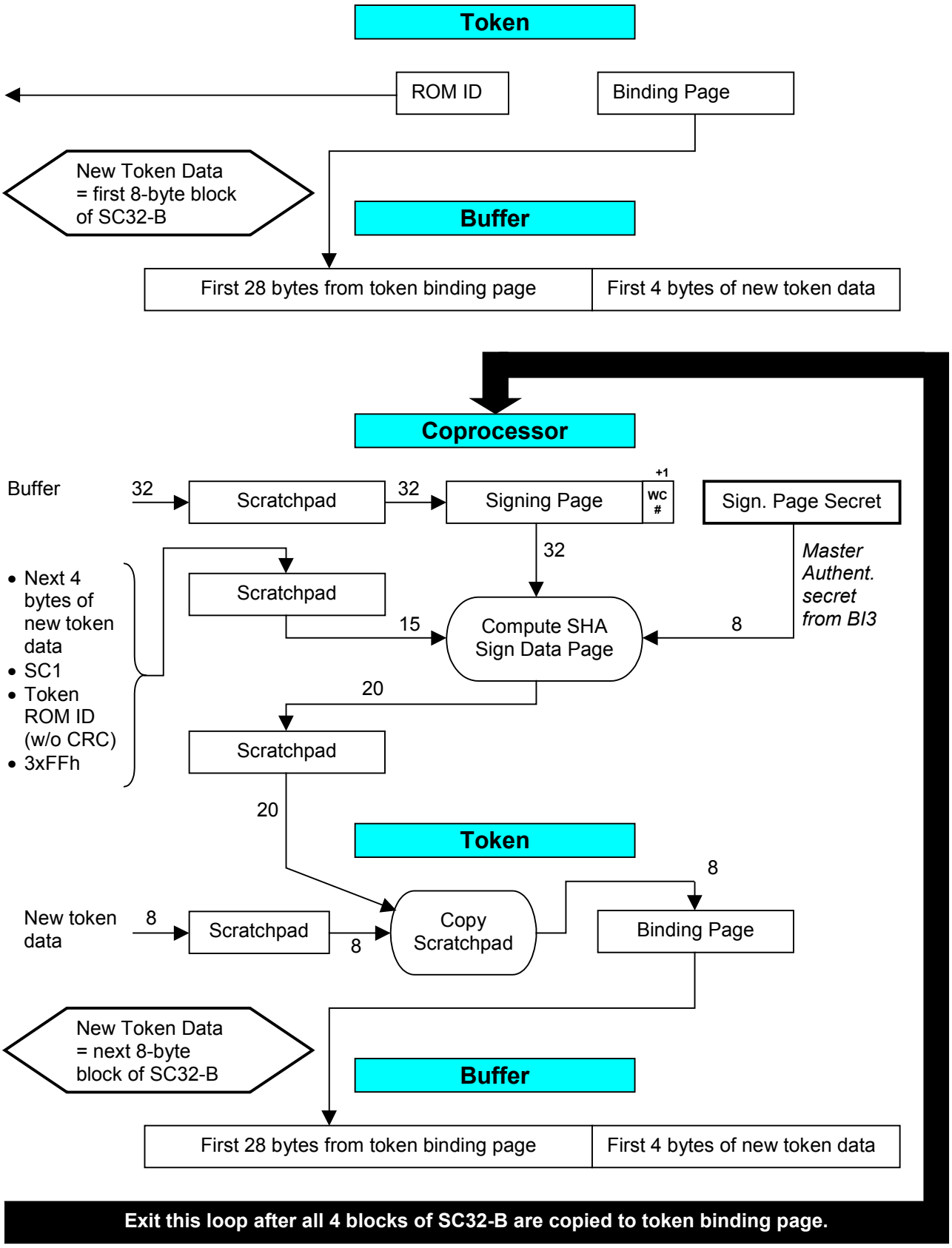
Step BI4

Title: Installation of the Unique Authentication Secret (UAS) in the token. This step is also called "Binding the secret to the token".

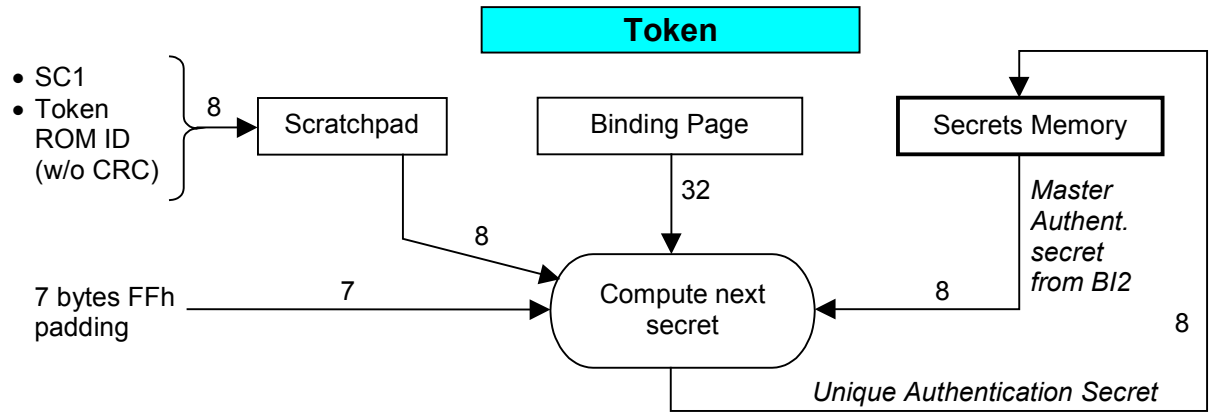
Precondition:
- BI2 was performed successfully.
- BI3 was performed successfully.
- The token is not write-protected.
- SC32-B is defined

Performed:
- Before installing a purse file in the token

Data Flow Diagram:



Data Flow Diagram (continued):



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Read ROM ID of token; verify correct reading
1)		Read data of token binding page; verify correct reading.
2)	Using a dummy starting address, erase the scratchpad.	
3)	For the new data to be written to the token use the first 8 bytes of SC32-B.	
A 4)	Using the starting address of the signing page, write the first 28 bytes of the token binding page to the scratchpad. For the remaining four bytes of the scratchpad use the first four bytes of the new token data.	
4)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
5)	Write to scratchpad locations 8 to 22: (data page address don't care) the next four bytes of the new token data, SC1 (token binding page number), token ROM ID (without CRC), 3 bytes FFh.	
6)	Using the starting address of the signing page issue Compute SHA/Sign Data Page. Now the scratchpad locations 8 to 27 contain the SHA-1 result needed to write the particular 8 bytes to the token.	
7)		Write the new token data to the token's scratchpad using the correct target starting address and verify correct scratchpad writing
7)		Issue the copy scratchpad command with the correct target address and a computed E/S byte and send the SHA-1 result computed by the co-processor.
8)		Read the full content of the binding page and store it in a buffer. Go to A and continue using the next 8-byte chunk of SC32-B until the whole SC32-B is written to the token binding page.
9)		Using the starting address of the token binding page, write to scratchpad: SC1, token ROM ID (without CRC).

Ref. #	DS1963S Coprocessor	DS1961S Token
10)		Issue the Compute Next Secret command with the correct starting address.

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare the computation of a SHA-1 result that is required by the token to copy scratchpad data to memory. *The ROM ID of the token and the data of the token's binding page are used as input for the SHA-1 computation.*
2)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
3)	To split the new data for the memory page into 8-byte chunks that fit into the scratchpad. *The size of the DS1961S scratchpad is eight bytes.*
4)	To prepare the computation of a SHA-1 result that is required by the token to copy scratchpad data to memory. *The signing page needs to hold the first 28 bytes of the token's memory page that is to be written to plus the first four new bytes for that memory page. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
5)	To load the scratchpad with the remaining 15 bytes that are needed by the coprocessor to compute a SHA-1 result that is required by the token to copy scratchpad data to memory. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
6)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor. It is required by the token to copy scratchpad data to memory.*
7)	To write one 8-byte segment of the new page data to the binding page in the token. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page. Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command. The execution of this command requires the SHA-1 result of Note 6).*
8)	To prepare the computation of the next SHA-1 result that is required by the token to copy scratchpad data to memory. *The memory data of the token's binding page is used as input to the SHA-1 computation. Since the page data has changed, the new data needs to be loaded for the SHA-1 computation (Note 4). Reading the data page allows the verification of successful execution of the copy scratchpad command.*
9)	To load the page number of the binding page and the token's ROM ID into the scratchpad. *This information is used to create the unique authentication secret.*
10)	To compute the unique authentication secret from the content of the binding page, token-specific scratchpad data and the current master authentication secret, and install it as the new secret in the token. *The target address must point to a location within the Binding Page. There is no need to erase the binding page in the token. Exposing the binding data does not compromise the system security since it is gives no clues on the Master Authentication Secret.*

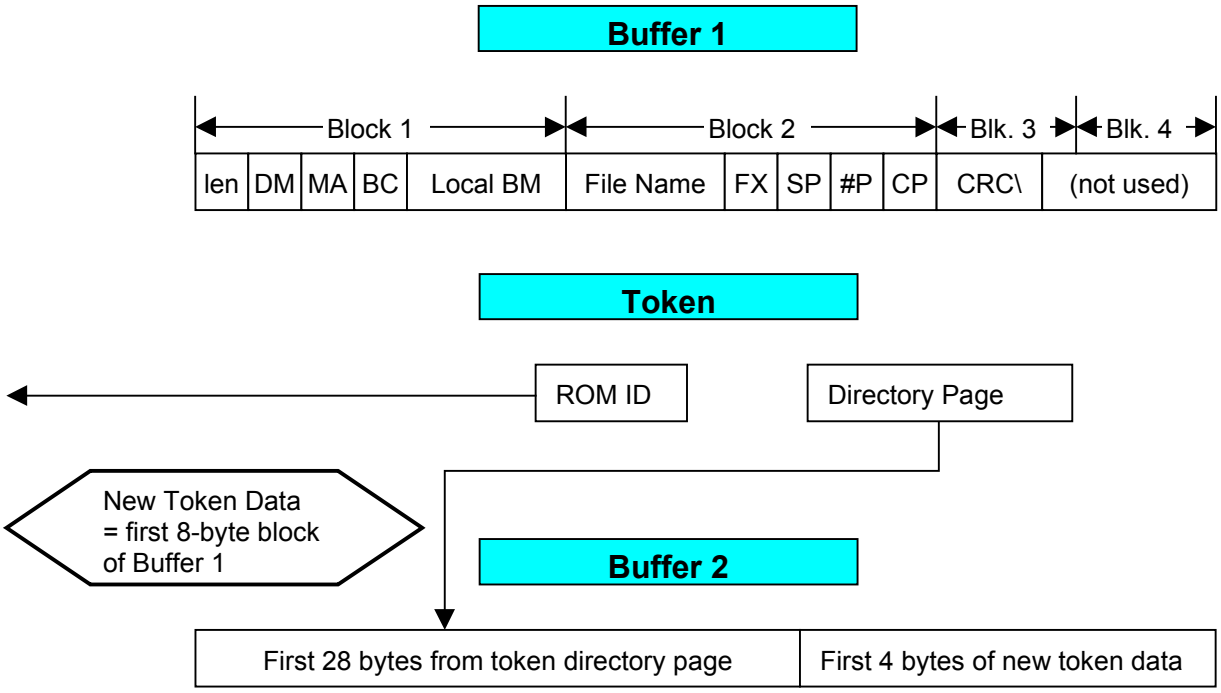
Step BI5

Title: Installation of a device file directory with an entry for the purse file in the token.

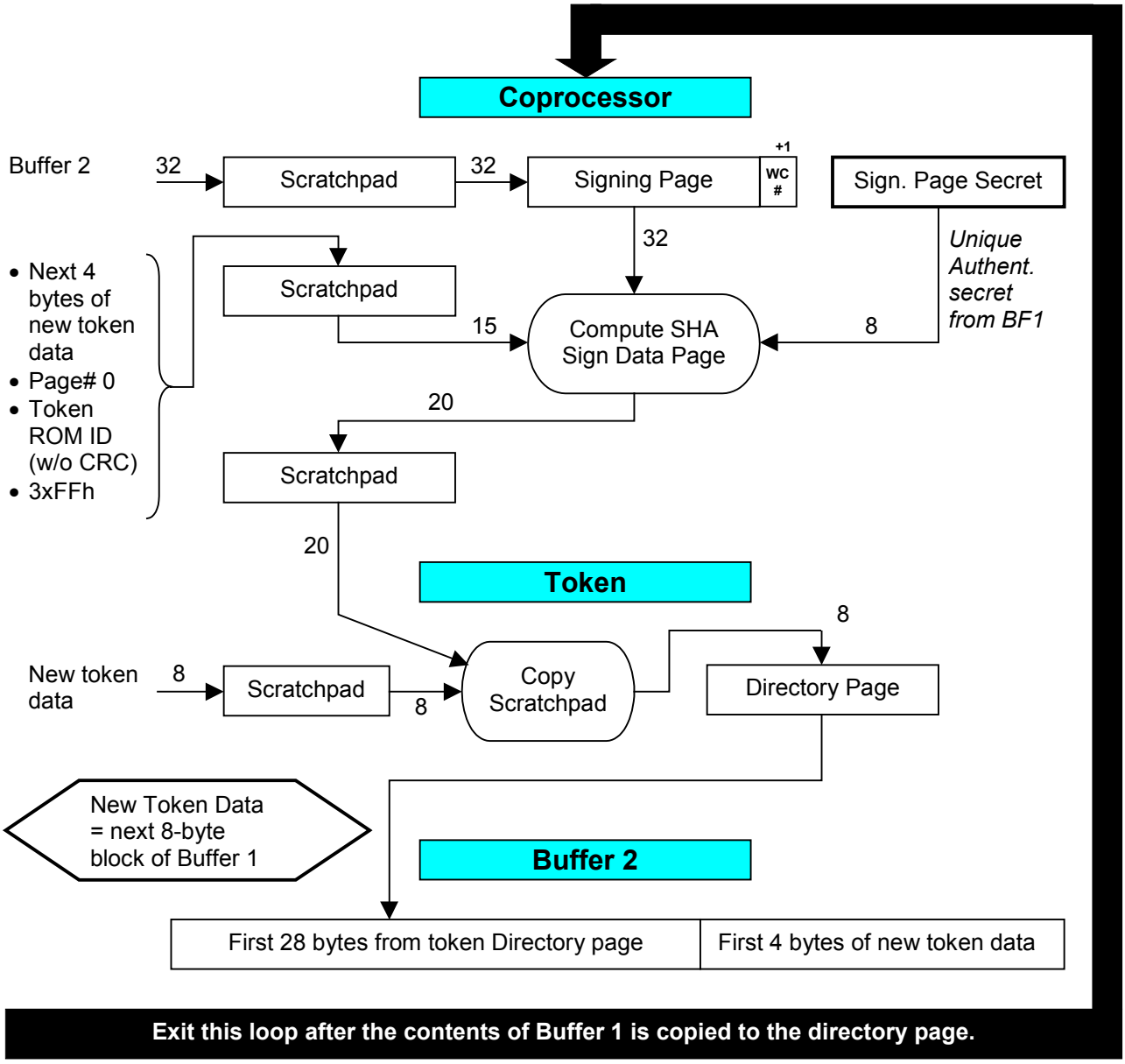
Precondition:
- BI4 was performed successfully.
- The file name of the purse file to be created is defined
- The page number (location) of the purse file to be created (length = 1 page or less) is defined

Performed:
- When initializing (commissioning) a token for use in the application.

Data Flow Diagram:



Data Flow Diagram (continued):



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)	Installation of the token's Unique Authentication Secret (UAS) as the secret of the workspace page and signing page of the coprocessor. (Same as step BF1)	
2)	Verify whether the token belongs to the system. (Essentially the same as step BF2; however, uses data of the directory page.)	
A 3)	Create the device directory with the entry of the purse file (name, page #, length) in the token, as shown in the data flow diagram.	
4)	Verify whether the token belongs to the system. (Essentially the same as step BF2; however, reads the directory page.)	
5)	Compare the directory page data read from the token to the expected data. If it is the same token as before, but the data doesn't match, go to A.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare for the verification of the token's unique authentication secret and for writing to the token. *If the token does not belong to the system, the verification will fail. The coprocessor needs to know the token's unique secret to write to the token.*
2)	To verify whether the secret in the token follows the rules that were defined for the application. *It is assumed that the unique authentication secret (UAS) has been installed in the coprocessor in step BF1. The verification works with any data page.*
3)	To install a device file directory in the Directory Page (page 0). *See Table 1 for the directory format. This data in buffer 1 assumes that the token is unused, i. e., no service record exists. Additional service records or purses may be installed later. The device supports a total of 3 service records that all share the same Unique Authentication Secret. When installing an additional purse or service record, the data for the directory page is created by first reading the existing directory and appending another file entry. See AN114 for the device directory format and definitions.*
4)	To verify whether token was not swapped after the file directory was written. *If the token was swapped, the verification will fail.*
5)	To verify whether the directory was installed properly. *If necessary, the directory installation is repeated.*

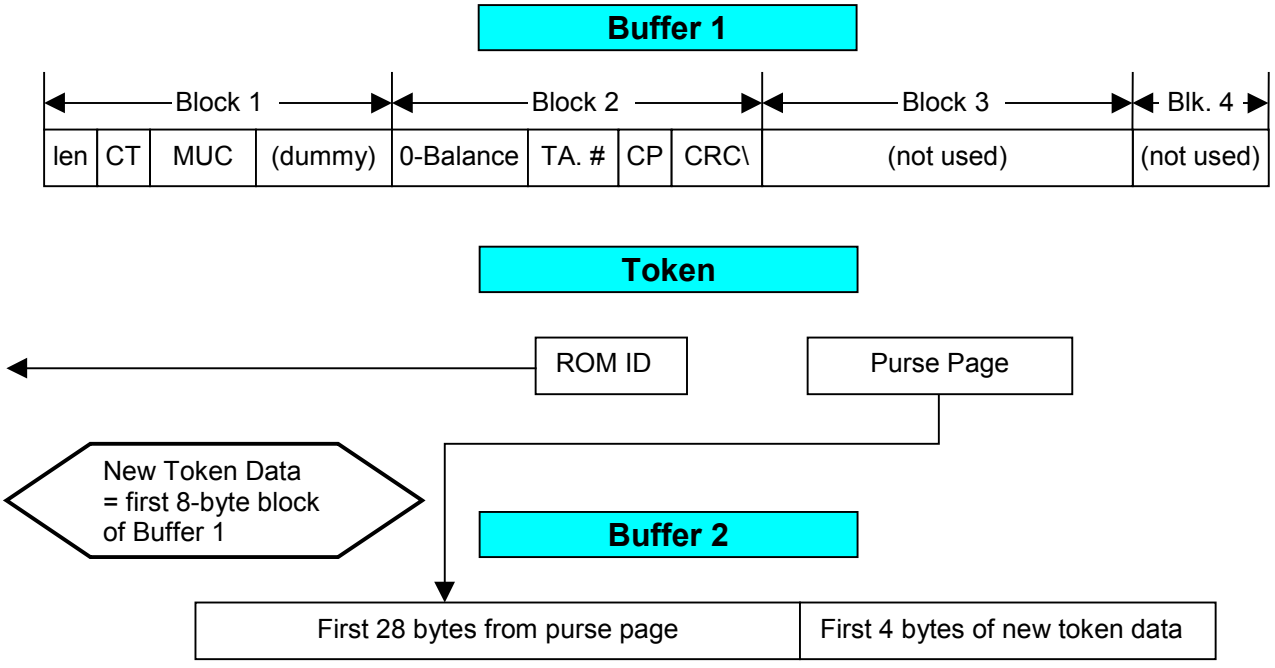
Step BI6

Title: Writing a zero-value purse file to the token

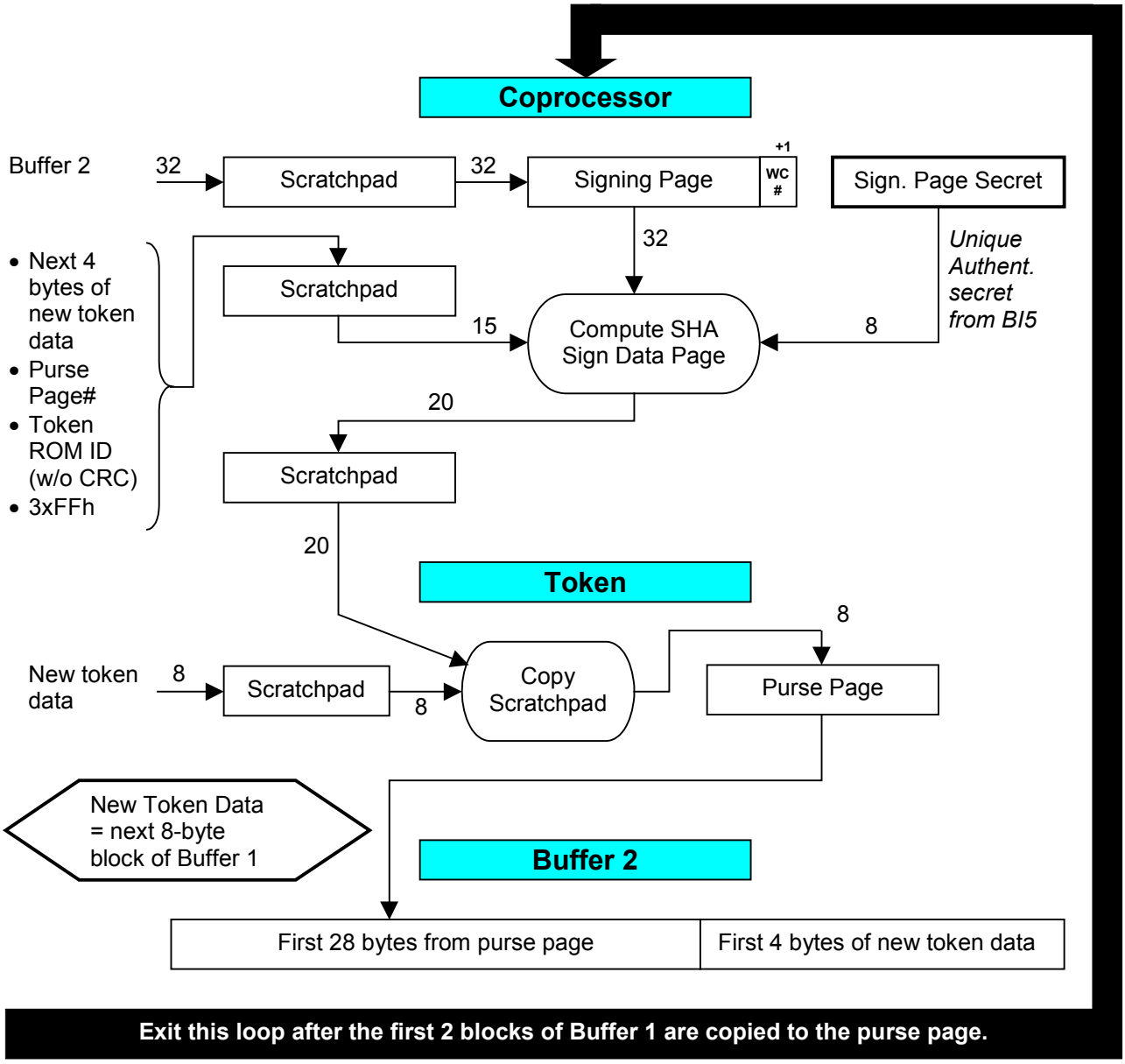
Precondition:
- BI5 was performed successfully immediately prior to this step.
- The location of the purse file is known (from BI5).
- The format and contents of the zero-value purse file are known.

Performed:
- Immediately before releasing the token for use in the application.

Data Flow Diagram:



Data Flow Diagram (continued):



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)	Verify whether the token belongs to the system. (Essentially the same as step BF2; the page number has to be the one of the new purse file.)	
A 2)	Create the zero-value purse file in the token, as shown in the data flow diagram.	
3)	Verify whether the token belongs to the system. (Essentially the same as step BF2; however, reads the new purse file.)	
4)	Compare the purse file read from the token to the expected data. If it is the same token as before, but the data doesn't match, go to A.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To verify whether the secret in the token follows the rules that were defined for the application. *It is assumed that the unique authentication secret (UAS) has been installed in the coprocessor in step BI5 (BF1). The verification works with any data page.*
2)	To install a valid (though empty) purse file in the particular token. *See Table 3 for the purse file format.*
3)	To verify whether token was not swapped after the purse file was installed. *If the token was swapped, the verification will fail.*
4)	To verify whether the purse file was installed properly. *If necessary, the purse file installation is repeated.*

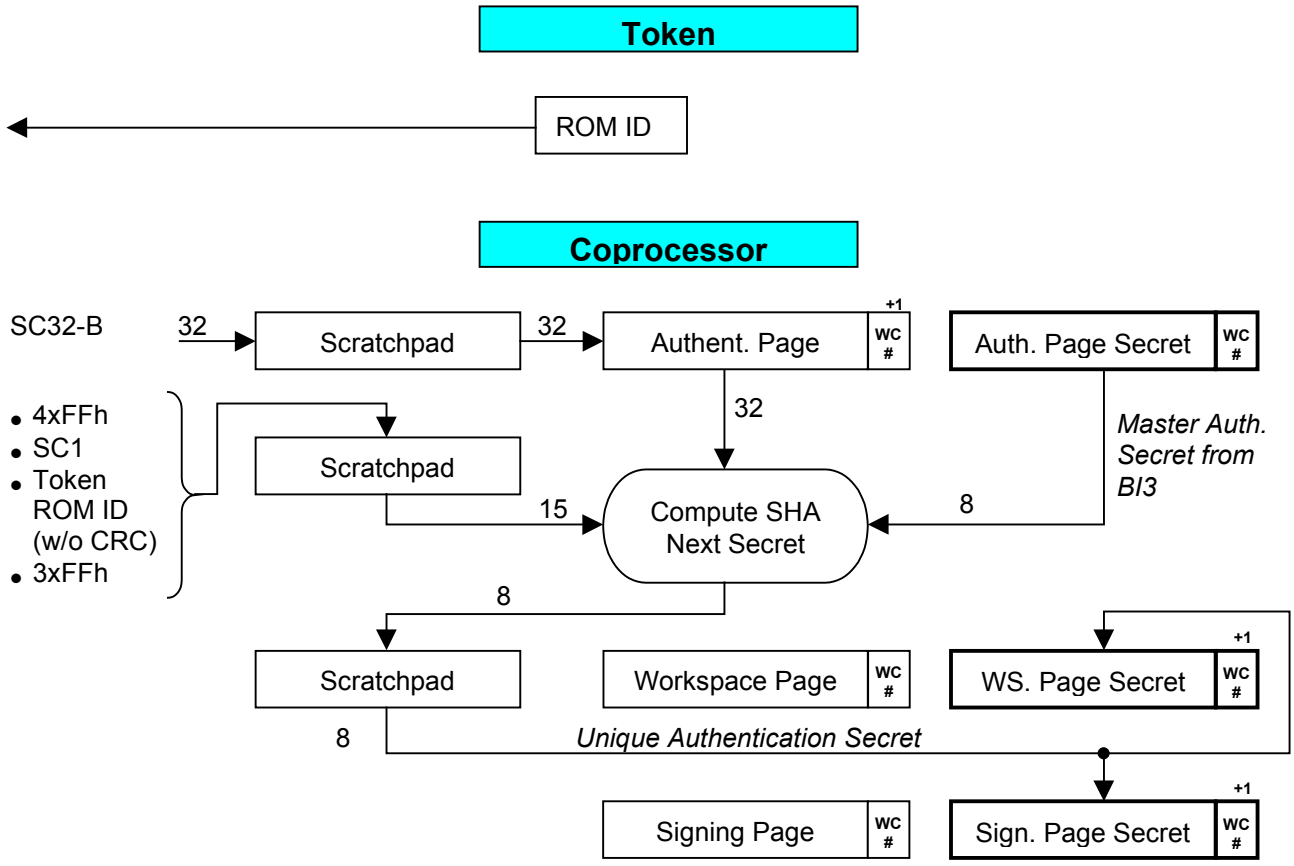
Step BF1

Task: Installation of the token's Unique Authentication Secret as the secret of the workspace page and signing page of the coprocessor

Precondition:
- BI3 was performed successfully on the coprocessor
- BI4 was performed successfully

Performed:
- Preparation for token authentication and updating the purse file

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Read ROM ID of token; verify correct reading.
2)	Using a dummy starting address, erase the scratchpad.	
3)	Using the starting address of the authentication page, write SC32-B to the scratchpad.	
3)	Verify correct scratchpad writing and copy scratchpad data to authentication page.	
4)	Write to scratchpad locations 8 to 22: (data page address don't care) four bytes FFh, SC1 (token binding page number), token ROM ID (without CRC), three bytes FFh.	
5)	Using the starting address of the authentication page issue Compute SHA/Next Secret.	
6)	Using the starting address of the secret of the workspace page, write 8 dummy bytes to the scratchpad.	
7)	Using the starting address of the secret of the workspace page and a computed E/S byte, issue the Copy Scratchpad command.	
8)	Using the starting address of the secret of the signing page, write eight dummy bytes to the scratchpad.	
9)	Using the starting address of the secret of the signing page and a computed E/S byte, issue the Copy Scratchpad command.	
2)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare computing the unique authentication secret of the particular token. *The ROM ID of the token is used as input to the SHA-1 computation.*
2)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*
3)	To load the 32 bytes Binding Data into the Authentication Page. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
4)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the token binding page number, and seven FFh padding bytes. *This data pattern is the same as the one used when installing the UAS in the token. The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
5)	To compute the unique authentication secret (UAS) of the particular token. *The target address must point to a location within the Authentication Page.*
6)	To select the secret of the Workspace Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Workspace Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
7)	To install the UAS as the secret of the Workspace Page. *Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*
8)	To select the secret of the Signing Page as the destination of the SHA-1 result. *The target address must point to a location within the secret of the Signing Page. At least one dummy byte needs to be sent to the scratchpad before issuing a reset pulse.*
9)	To install the UAS as the secret of the Signing Page. *The SHA-1 result is still hidden in the scratchpad. Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command.*

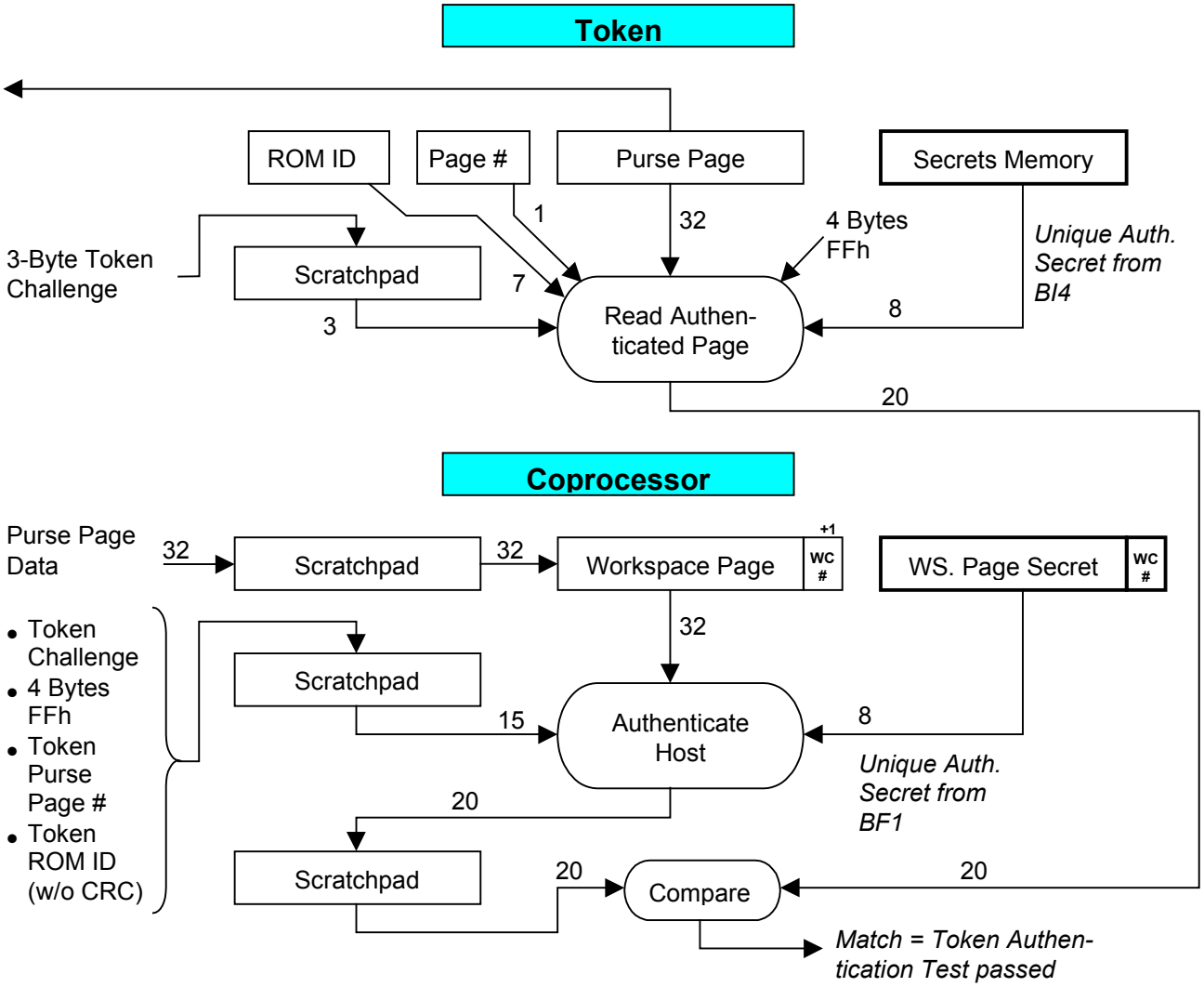
Step BF2

Title: Verify whether the token belongs to the system

Precondition:
- BF1 was performed successfully immediately prior to this step
- BI6 or BF3 was performed successfully

Performed:
- Before updating the purse file

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Write an 8-byte random number to scratchpad. (The page address used is not relevant.) The content of scratchpad locations 4 to 6 will be used as challenge with the next command.

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Perform a Read Authenticated Page Command using the starting address of the page that contains the purse file. Error-check and save the page data in a buffer. Error-check and save the SHA-1 result in a buffer.
2)	Using the starting address of the workspace page, write the page data read from the token to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to workspace page.	
3)	Write to scratchpad locations 8 to 22: (data page address don't care) 4 bytes FFh, token purse page number, token ROM ID (without CRC), the same random number that was used with Read Authenticated Page.	
4)	Using the starting address of the workspace page issue the Authenticate Host command.	
4)	Take the SHA-1 result from the Read Authenticated Page command and use it with the Match Scratchpad command. If this command results in AAh pattern, the SHA results did match, confirming that the token belongs to the system.	
5)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To read the purse page and to obtain a SHA-1 result from the token based on a 3-byte challenge, purse page data, token ROM ID, purse page number, and UAS of the token. *The scratchpad of the DS1961S can only be written in 8-byte blocks. The first four and the last byte written to the scratchpad are not used for this SHA-1 computation. Using a random challenge generates different SHA-1 results from otherwise identical input data. Only the legitimate token can perform the correct SHA-1 computation.*
2)	To load the 32 bytes purse file into the Workspace Page. *This is a precondition for re-computing the SHA-1 result that was read from the token. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the challenge that was used when reading the purse, and four FFh padding bytes. *This is a precondition for recomputing the SHA-1 result that was read from the token. The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
4)	To recompute the SHA-1 result that was read from the token. To compare the SHA-1 result from the token to the one computed in the coprocessor. *The target address must point to a location within the Workspace Page. To verify the authenticity of the DS1961S token, the Authenticate Host command must be used. The reason for this is in the pattern "01" for the upper bits of the MP byte that is used with the Read Authenticated Page command. To set the same conditions with the DS1963S for verification, the upper two bits of the MPX byte need to have the same pattern. This is only the case with the Authenticate Host command. If both SHA-1 results match, the secret of the token was computed according to the rules that were defined for the application. This is the evidence that the token belongs to the system.*
5)	To clear the HIDE flag, which, when cleared, opens the scratchpad for write access. To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*

Step BF3

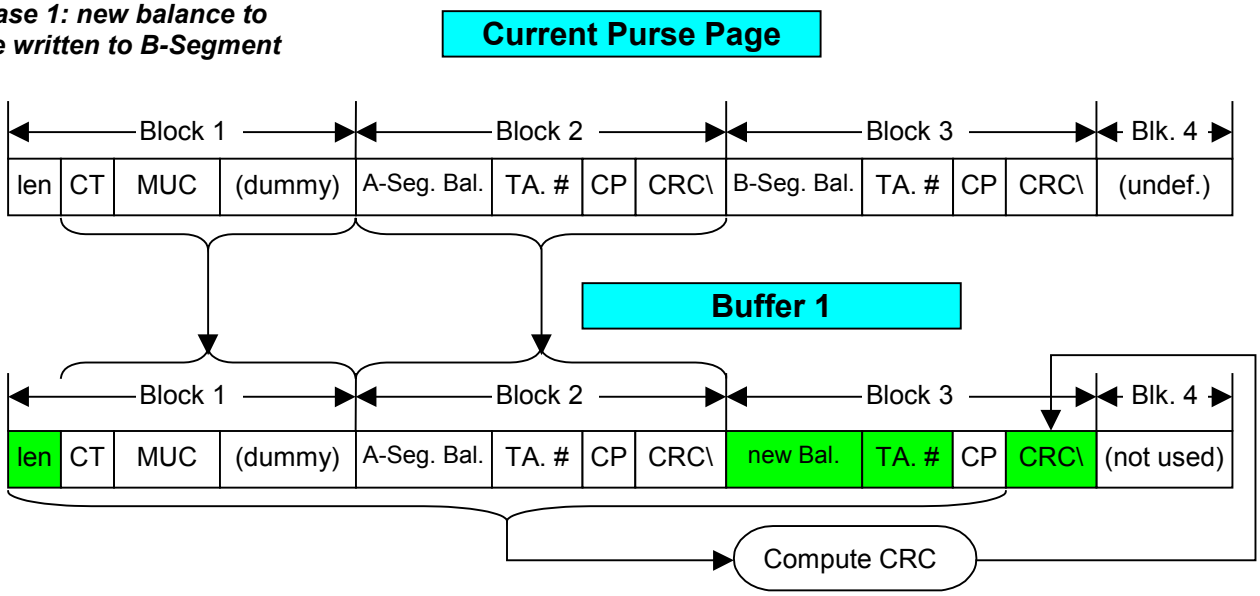
Title: Updating the purse file in the token

Precondition:
- BF2 was performed successfully
- The new data for the purse file has been determined

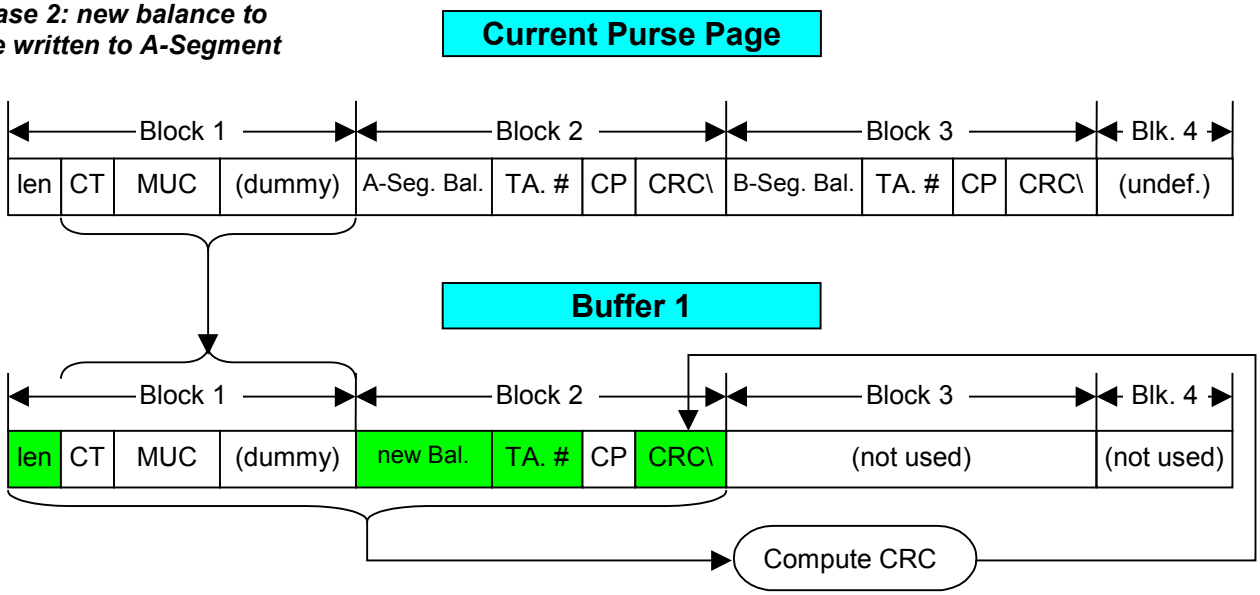
Performed:
- Precondition for releasing the purchased goods

Data Flow Diagram:

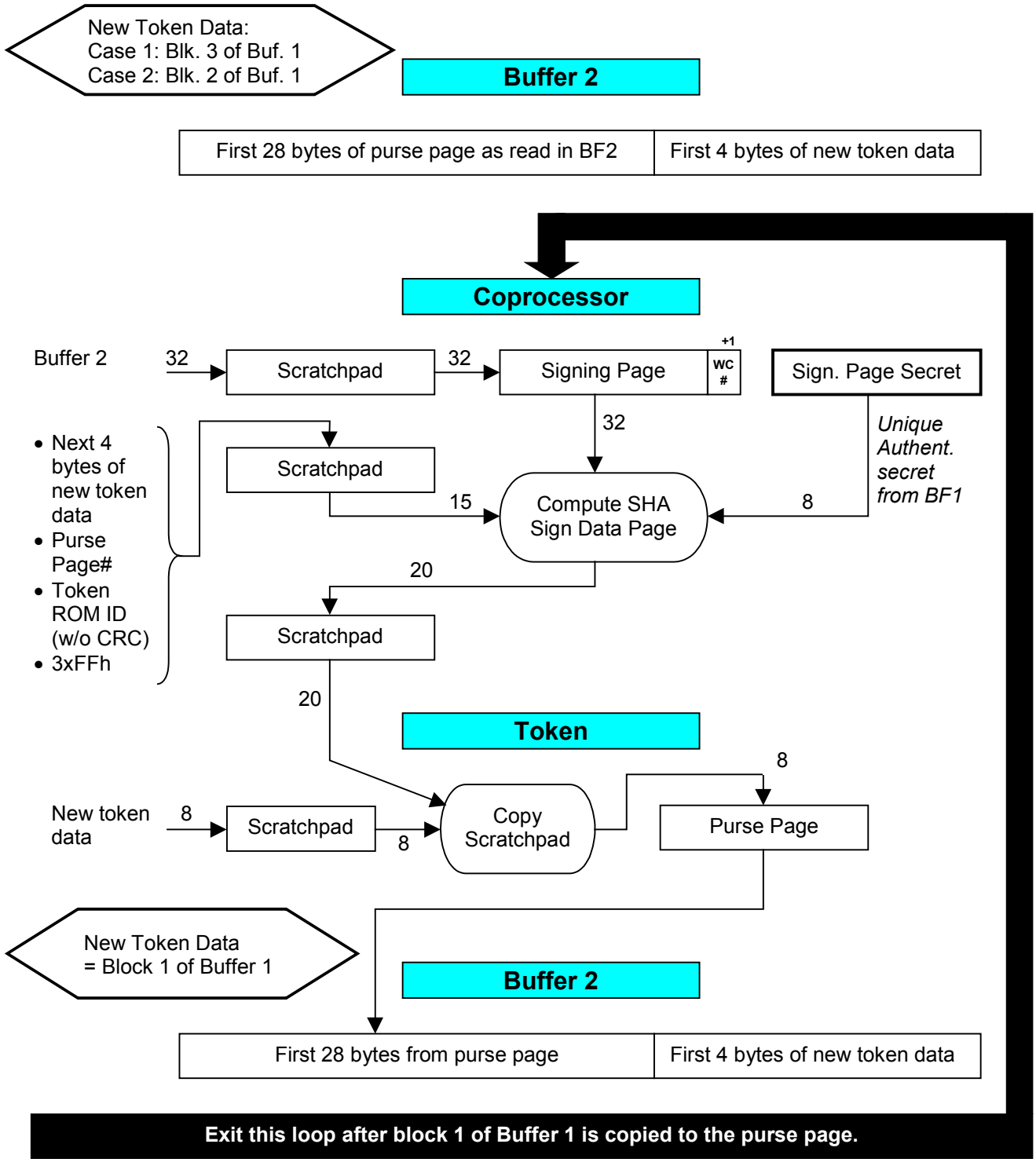
Case 1: new balance to be written to B-Segment



Case 2: new balance to be written to A-Segment



Data Flow Diagram (continued):



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
A 1)	Using the starting address of the signing page, write the first 28 bytes of the token purse page data to the scratchpad. For the remaining four bytes of the scratchpad use the first four bytes of the new token data.	

Ref. #	DS1963S Coprocessor	DS1961S Token
1)	Verify correct scratchpad writing and copy scratchpad data to signing page.	
2)	Write to scratchpad locations 8 to 22: (data page address don't care) the next four bytes of the new token data, token purse page number, token ROM ID (without CRC), three bytes FFh.	
3)	Using the starting address of the signing page issue Compute SHA/Sign Data Page. Now the scratchpad locations 8 to 27 contain the SHA-1 result needed to write the particular eight bytes to the token.	
4)		Write the new token data to the token's scratchpad using the correct target starting address and verify correct scratchpad writing
4)		Issue the copy scratchpad command with the correct target address and a computed E/S byte and send the SHA-1 result computed by the co-processor.
5)		After having updated the A- or B-segment of the purse file, read the full content of the purse page and store it in a buffer. Go to A and continue using block 1 of buffer 1 as the new token data until the update of the purse file is completed.
6)	Using a dummy starting address, erase the scratchpad.	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To prepare the computation of a SHA-1 result that is required by the token to copy scratchpad data to memory. *The signing page needs to hold the first 28 bytes of the token's memory page that is to be written to plus the first four new bytes for that memory page. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
2)	To load the scratchpad with the remaining 15 bytes that are needed by the coprocessor to compute a SHA-1 result that is required by the token to copy scratchpad data to memory. *The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*
3)	To compute a SHA-1 result from the content of the Signing Page, 15 bytes of known scratchpad data and the secret of the signing page. *The target address must point to a location within the Signing Page. The SHA-1 result will be found in the scratchpad of the coprocessor. It is required by the token to copy scratchpad data to memory.*
4)	To write one 8-byte segment of the new page data to the purse page in the token. *Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page. Target address and E/S byte could as well be obtained from first reading the scratchpad before issuing the copy scratchpad command. The execution of this command requires the SHA-1 result of Note 3).*
5)	To prepare the computation of the next SHA-1 result that is required by the token to copy scratchpad data to memory. *The memory data of the token's purse page is used as input to the SHA-1 computation. Since the page data has changed, the new data needs to be loaded for the SHA-1 computation (Note 1). Reading the data page allows the verification of successful execution of the copy scratchpad command.*
6)	To erase data in the scratchpad. *The target address issued is not relevant; any value is accepted.*

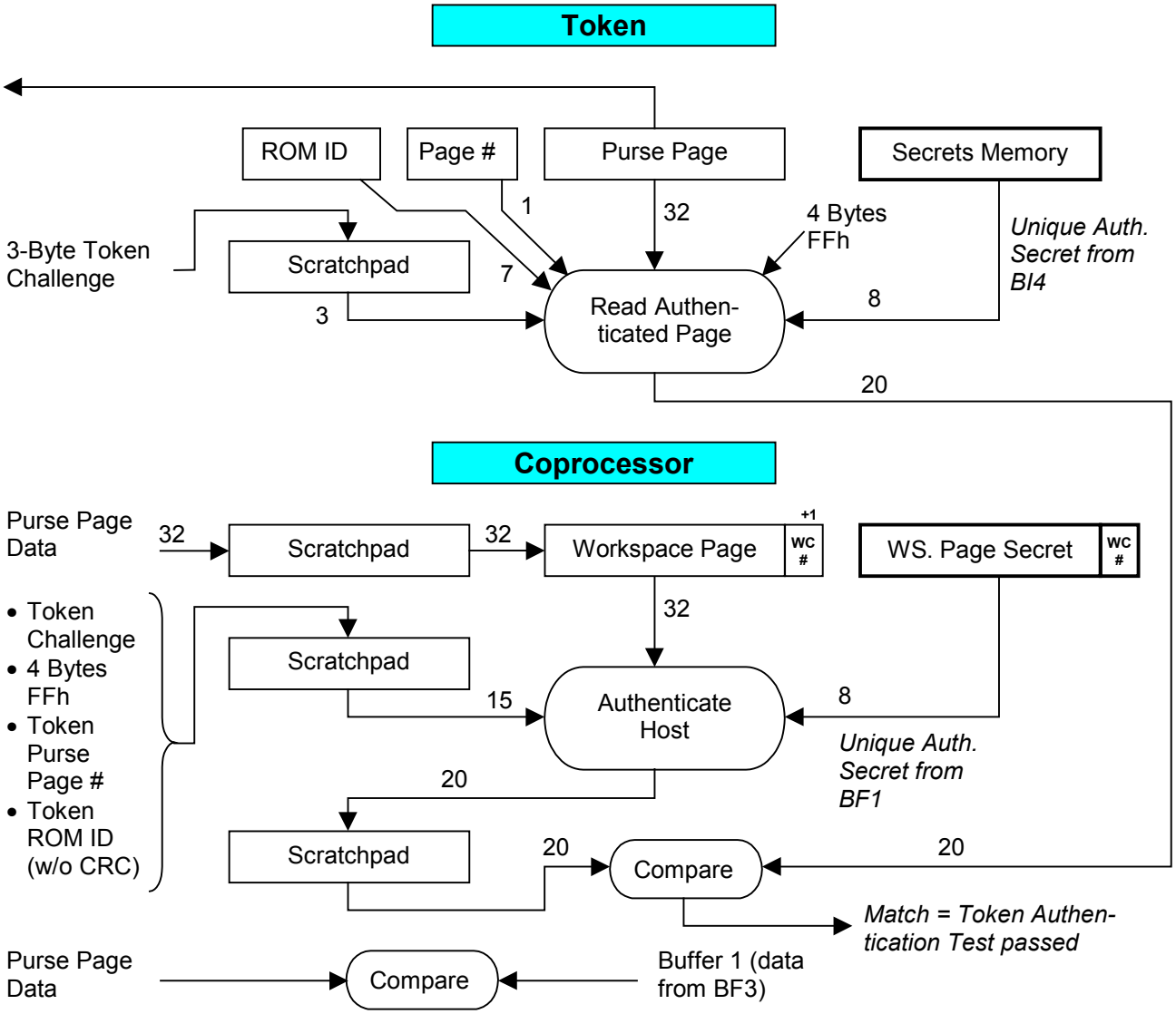
Step BF4

Title: Verify whether the purse file was written successfully to the same token that was read before.

Precondition:
- BF3 was performed successfully
- The expected content of the purse file is known.

Performed:
- Immediately before dispensing goods

Data Flow Diagram:



Detail Description:

Ref. #	DS1963S Coprocessor	DS1961S Token
1)		Write an 8-byte random number to scratchpad. (The page address used is not relevant.) The content of scratchpad locations 4 to 6 will be used as challenge with the next command.
1)		Perform a Read Authenticated Page Command using the starting address of the page that contains the purse file. Error-check and save the page data in a buffer. Error-check and save the SHA-1 result in a buffer.
2)	Using the starting address of the workspace page, write the page data read from the token to the scratchpad.	
2)	Verify correct scratchpad writing and copy scratchpad data to workspace page.	
3)	Write to scratchpad locations 8 to 22: (data page address don't care) 4 bytes FFh, token purse page number, token ROM ID (without CRC), the same random number that was used with Read Authenticated Page.	
4)	Using the starting address of the workspace page issue the Authenticate Host command.	
4)	Take the SHA-1 result from the Read Authenti-cated Page command and use it with the Match Scratchpad command. If this command results in AAh pattern, the SHA results did match, con-firming that the token belongs to the system and that it is the same token as in previous steps.	
5)	Compare the data read from the token to the expected purse file data. If it is the same token as before, but the data doesn't match, repeat step BF3 (or BI6, respectively).	

Detail Notes:

Ref. #	Purpose and *Comments*
1)	To read the purse page and to obtain a SHA-1 result from the token based on a 3-byte challenge, purse page data, token ROM ID, purse page number, and UAS of the token. *The scratchpad of the DS1961S can only be written in 8-byte blocks. The first four and the last byte written to the scratchpad are not used for this SHA-1 computation. Using a random challenge generates different SHA-1 results from otherwise identical input data. Only the legitimate token can perform the correct SHA-1 computation.*
2)	To load the 32 bytes purse file into the Workspace Page. *This is a precondition for re-computing the SHA-1 result that was read from the token. Data is first written to the scratchpad, verified (e.g., read back) and then copied to the memory page.*
3)	To load the scratchpad with 15 bytes of data that identify the token (ROM ID), the memory page used for the purse, the challenge that was used when reading the purse, and four FFh padding bytes. *This is a precondition for recomputing the SHA-1 result that was read from the token. The target address must point to byte number 8 of any regular memory page (T15:T0 = 0000 000x xxx0 1000b).*

Ref. #	Purpose and *Comments*
4)	To recompute the SHA-1 result that was read from the token. To compare the SHA-1 result from the token to the one computed in the coprocessor. *The target address must point to a location within the Workspace Page. To verify the authenticity of the DS1961S token, the Authenticate Host command must be used. The reason for this is in the pattern "01" for the upper bits of the MP byte that is used with the Read Authenticated Page command. To set the same conditions with the DS1963S for verification, the upper two bits of the MPX byte need to have the same pattern. This is only the case with the Authenticate Host command. If both SHA-1 results match, the secret of the token was computed according to the rules that were defined for the application. This is the evidence that the token belongs to the system.*
5)	To verify whether the updated purse file (after-purchase value) was actually written to the token. *From the previous step (BF3) the host still knows the expected new purse file data. Only if the purse was updated will the merchandise be released.*

APPENDIX C, SECURITY Q&A

DS1963S Token

What can happen if an attacker knows the authentication secret of a purse?
Knowing the authentication secret, an attacker can program a microcontroller to emulate (behave like) a token in all of its functions. The emulated device will pass the authentication test (because it knows the secret) and the purse file will be accepted (because the embedded signature was copied from a valid device). After purchases the emulator can be reset to start over again at the original purse value, this way spending the initially loaded amount of money multiple times.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. A system that uses unique authentication secrets limits the attacker's success to only one part. In a system that uses common authentication secrets (same secret for all purses) an attacker who knows the authentication secret could emulate all tokens, forcing the service provider to install new secrets in all tokens and updating the secrets in all vending and revaluing stations.

How feasible is it for an attacker to discover an authentication secret?
If the secret is actually computed rather than loaded like a password, it is virtually impossible to guess it of discover it by trial and error. The secret's size of 64 bits allows for 2^{64} or $18.446744 * 10^{18}$ possible combinations. Using a fast computer that can test 1 million tests per second, testing all combinations will take no longer than $18.446744 * 10^{12}$ seconds. With 60*60*24 or 86400 seconds per day, this takes $213.5 * 10^{6}$ days or 580000 years. The security of the secret is maximized by the fact that it can only be changed all 64 bits at a time. If the secret could be changed one bit at a time, an attacker could discover it after maximum 64 attempts. If the secret could be changed one byte at a time, an attacker could discover it after maximum 8*256 or 2048 attempts. The 64-bits at a time concept, however, creates the maximum possible hurdle for an attacker to discover the secret. Even if one secret was discovered, only a single device could be emulated --- provided that the application uses unique authentication secrets.

What can happen if an attacker knows how to create valid authentication secrets?
To create valid authentication secrets an attacker needs to know the Authentication Input Secret (SC32-1, SC15-1), Binding Data (SC32-B), Binding Code (SC7-B) and public data pertaining to the particular token. As with all secrets, the service provider must ensure that the secret information does not get into the wrong hands. With humans being the weakest link in the chain, the best way to achieve the desired security is by means of partial secrets (SC32-2, SC15-2, SC32-3, SC15-3, etc.). Even if these parameters leaked out and an attacker would install valid authentication secrets in legally obtained tokens, these devices will not be able to cause any damage because the purses will be rejected due to invalid signature.

What can happen if an attacker knows how to create valid purse signatures?
To create valid purse signatures an attacker needs to know the Signing Input Secret (SC32-S, SC15-S), Initial Signature (SC20-S), Signing Challenge (SC3-S), and public data pertaining to the particular token. The service provider must ensure that the secret information does not get into the wrong hands. Although not mentioned in the Scenario A description, the signing secret can be computed from partial secrets the same way as the master authentication secret is computed. Knowing how to create valid signatures, an attacker could revalue any purse that has a valid authentication secret. This would force the service provider to change the system-wide Signing Secret. Using fresh tokens without a valid authentication secret, the system cannot be attacked because these parts would not pass the authentication test.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. Develop and implement a plan to rotate the signing secret regularly.

Why is page 8 not recommended for purse files?
If a purse were installed in page 8, it may be possible to attack the system through replay of a valid purse file. In this case there is no need for the attacker to know any secret. This attack could work with any valid token that has money in the purse file. As a precondition for this attack one needs a valid token that is connected to a fast microcontroller. On the other side of the microcontroller is an empty iButton, which (instead of the token) gets in contact with the vending station. The microcontroller memorizes the original purse content and the page write cycle count that the purse's signature is valid with.
When during a transaction (Step AF2) the vending station issues the Read Authenticated Page command, the microcontroller writes the saved counter value, purse page number, token ROM ID and challenge to the token's scratchpad. Instead of the Read Authenticated Page command, the micro issues the Sign Data Page command. To the host the micro replays the saved purse content, saved cycle counter and then transmits the SHA-1 result from Sign Data Page. The host will accept the data for authentication and purse signature test. When the vending station updates the purse (Step AF4), the microcontroller will let the communication pass through to the token. When the token and the new data are verified by the vending station (Step AF5), the microcontroller writes the incremented saved counter value, purse page number, token ROM ID and challenge to the token's scratchpad. Instead of the Read Authenticated Page command, the microcontroller issues the Sign Data Page command. To the vending station the micro replays the new purse data, incremented saved cycle counter and then transmits the SHA-1 result from Sign Data Page.
Remedy: This type of attack can only succeed if the vending station cannot detect any delays in the communication with the token. A vending station operating at standard 1-Wire speed and a token communicating with the microcontroller at Overdrive speed makes this attack possible. Therefore, always communicate with the token at Overdrive speed and do not use page 8 for a purse file.

What can happen if an attacker cracks open a vending station and gets physical access to the inside of the electronics box?
If the electronics box remains functioning, the attacker could eavesdrop on the communication between microcontroller and coprocessor during a vending transaction. During the initial vending step AF1, the Binding Data (SC32-B) and Binding Code (SC7-B) are exposed as they are written to the coprocessor. When the purse's signature is verified in step AF3, the Initial Signature SC20-S and the Signing Challenge SC3-S are exposed on their way to the coprocessor. Knowing these system constants the attacker has two choices: a) take the coprocessor and use it to revalue purses (see *What can happen if an attacker has a coprocessor that is set up for an application? Part 1)* or b) convert the coprocessor into a token for purse replay attacks (see *What can happen if an attacker has a coprocessor that is set up for an application? Part 2)*.

What can happen if an attacker has a coprocessor that is set up for an application? Part 1
Having access to a coprocessor with a valid signing secret and knowing SC20-S and SC3-S, the attacker can revalue any purse that is part of the system, i. e., which passes the authentication test.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. Develop and implement a plan to rotate the signing secret regularly. Keep the coprocessor and system control unit locked in a physically secure enclosure that cannot be opened without major damage to its contents.

What can happen if an attacker has a coprocessor that is set up for an application? Part 2
Having access to a coprocessor with a valid Master Authentication Secret and knowing SC32-B and SC7-B, the attacker can make the coprocessor compute the SHA-1 result for any DS1963S ROM ID to pass the authentication test of step AF2. To take advantage of this capability the attacker needs to also have access to a valid token that belongs to the system. The attack uses a similar technique as described in *Why is page 8 not recommended for purse files?* To prepare the attack, the purse file of a valid token is first copied to page 8 of the stolen coprocessor. Next the Unique Authentication Secret of the valid token is installed as the secret of page 8. This procedure is very similar to step AF1, however, the secret is now loaded to starting address 0200h, the location of the Master Signing Secret. For this reason the coprocessor loses its capability to compute signatures for purses.

The prepared coprocessor is connected to a fast microcontroller. On the other side of the microcontroller is an empty iButton, which (instead of a token) gets in contact with the vending station. The microcontroller needs to memorize the original purse data, the page write cycle count that the purse's signature is valid with, and the ROM ID of the token that the purse file was copied from. Whenever during a vending transaction the command for reading the token's ROM ID appears (e. g., step AF1), the microcontroller replays the ROM ID of the valid token. At step AF2 a Read Authenticated Page command needs to be performed. To generate a proper response, the microcontroller writes the original purse write cycle counter value, original purse page number, token ROM ID and challenge to the coprocessor's scratchpad. Next the microcontroller changes the purse page address to page 8, and instead of Read Authenticated Page issues the Sign Data Page command. To the vending station the micro replays the saved purse data, saved cycle counter and then transmits the SHA-1 result from Sign Data Page. The vending machine will accept the data for authentication and purse signature test.

When the vending station updates the purse (Step AF4), the microcontroller changes the purse page address to page 8 but otherwise lets the communication pass through to the stolen coprocessor. When the token and the new data are verified by the vending station (Step AF5), the microcontroller writes the incremented original counter value, original purse page number, token ROM ID and challenge to the

coprocessor's scratchpad. Next the microcontroller changes the purse page address to page 8, and instead of Read Authenticated Page issues the Sign Data Page command. To the vending station the micro replays the new purse data, incremented original cycle counter and then transmits the SHA-1 result from Sign Data Page. This replay attack will be successful until the mismatch between revaluing and spending is discovered and the ROM ID of the emulated token is blacklisted.
Remedy: This type of attack is only possible if the vending station communicates at standard speed and the microcontroller communicates with the converted coprocessor at Overdrive speed. Therefore, vending stations should always communicate with the tokens at Overdrive speed.

What can happen if an attacker has two or more coprocessors of the same system?
If an attacker also knows the system constants SC32-B, SC7-B, SC20-S and SC3-S, an attack is possible that uses one coprocessor to create valid purse files for invented DS1963S ROM IDs and another coprocessor to replay these files as described in *What can happen if an attacker has a coprocessor that is set up for an application? Part 2*. Creating a new purse file for a new ROM ID after each purchase makes blacklisting a futile effort since the same ROM ID needs never appear again.
Remedy: This type of attack is only possible if the vending station communicates at standard speed and the microcontroller communicates with the converted coprocessor at Overdrive speed. Therefore, vending stations should always communicate with the tokens at Overdrive speed. Since it is impractical to store the ROM IDs of all valid tokens in all vending and revaluing stations or to always dial-in to a central database to verify a ROM ID, it is crucial to prevent an attacker from discovering the system constants when they are exposed inside the electronics box. To protect these constants, a vending station could be equipped with sensors that detect physical intrusion and in the event instruct the host processor to erase the secrets from the coprocessor. In addition, after each power-up, the vending stations could dial-in to the service provider and report the power-down event. The vending station could refuse any user transactions unless the contact with the service provider was established. If available, the service provider could use "Caller ID" to verify the phone number the vending station is calling from. A phone number that is not listed in the database indicates that the vending station resides at an unknown place. In that case the vending station could be instructed to erase the secrets from its coprocessor.

Why should I use a coprocessor if the microcontroller in the system control unit is fast enough to perform the SHA-1 computations?
To keep the input secrets from getting into the wrong hands, the microcontroller needs to be secure, i. e., it must be a unit that can be locked to prevent reading and disassembling the firmware. A secure microcontroller is quite expensive and may be slower in the SHA-1 computation than a DS1963S coprocessor. The coprocessor approach allows using a standard microcontroller, reducing the overall hardware cost. In addition, the coprocessor supports the use of partial secrets. With a secure microcontroller the secrets are known and accessible at the time of programming.

Why is data for DS1963S SHA commands loaded into the scratchpad starting at address 8 or 20?
The SHA-1 result that the DS1963S generates is loaded into the scratchpad starting at address 8. This was done to allow space at the beginning and the end of the scratchpad for TMEX formatting. For simplicity, this starting address was then used for other commands that require input through the scratchpad. Except for Read Authenticated Page and Compute Challenge, the scratchpad is loaded with 15 bytes of data. The first 12 of these bytes were originally intended to accommodate the purse page write cycle counter (4 bytes), token data page number (MPX byte), and the token's ROM ID without CRC, as used with the Sign Data Page command. The 3-byte challenge was just appended to that string. The same location of the challenge was then used with Read Authenticated Page and Compute Challenge.

What are the mysteries about the M-bit, X-bit and SEC#?
The "M-bit" of the DS1963S is intended for use with the user-authentication scheme, which is not recommended because of its inherent weakness.
The "X-bit" of the DS1963S is set only with the commands Compute Challenge and Authenticate Host. These commands are intended for use with the user-authentication scheme. The X-bit being set makes the SHA-1 result different from what one gets with other commands if all the other input data were the same.
The "SEC#" of the DS1963S was also intended for use with the user-authentication. The default value of SEC# is 000; it is loaded with the page number when using the Compute Challenge command. So far the SEC# is only used in the application "small cash with DS1961S token" when verifying the authenticity of the token using the Authenticate Host command. The Authenticate Host command needs to be used because the DS1961S sets the X-bit to 1 when performing the Read Authenticated Page command. With a DS1963S token, the Validate Data Page command is used for that purpose.

DS1961S Token

What can happen if an attacker knows the authentication secret of a token?
Knowing the authentication secret, an attacker can program a microcontroller to emulate (behave like) a token in all of its functions. The emulated device will pass the authentication test (because it knows the secret). The purse file will be accepted, since it does not contain any signature. After purchases the emulator can be reset to start over again at the original purse value, this way spending the initially loaded amount of money multiple times. Instead of using an emulator, the attacker could as well restore the purse file to its original value.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. A system that uses unique authentication secrets limits the attacker's success to only one part. In a system that uses common authentication secrets (same secret for all tokens) an attacker who knows the authentication secret could load the secret into fresh tokens (not requiring an emulator) and install valid purse files. This would force the service provider to install new secrets in all tokens and updating the secrets in all vending and revaluing stations. Therefore, in applications that use the DS1961S as a token, the use of unique authentication secrets is an *absolute requirement*.

How feasible is it for an attacker to discover an authentication secret?
(Same as with DS1963S token)

What can happen if an attacker knows how to create valid authentication secrets?
To create valid authentication secrets an attacker needs to know the Authentication Input Secret (SC32-1, SC8-1), Binding Data (SC32-B), Binding Page Number (SC1) and public data pertaining to the particular token. The service provider must ensure that the secret information does not get into the wrong hands. The best way to achieve the desired security is by means of partial secrets (SC32-2, SC8-2, SC32-3, SC8-3, etc.). Knowing all this secret information, an attacker could install valid authentication secrets and valid purse files in fresh tokens as well as re-value purses.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. Develop and implement a plan to rotate the authentication secret regularly.

What can happen if an attacker gets hold of a coprocessor that is set up for an application?
Having access to a coprocessor with a valid Master Authentication secret and knowing Binding Data (SC32-B), Binding Page Number (SC1), and public data pertaining to the particular token, the attacker can revalue any purse that is part of the system. However, the attacker cannot set up fresh tokens because this requires also knowledge of the Authentication Input Secret. SC32-B and SC1 can be obtained by eavesdropping on the communication with a coprocessor in a vending or revaluing station.
Remedy: Set up the revaluing and vending stations to keep transaction records and upload the records into a central account database. Comparing revaluing and spending history for each token identified by its ROM ID reveals fraudulent activity. Install a blacklist of tokens to be rejected in all vending stations. Develop and implement a plan to rotate the authentication secret regularly. Keep the coprocessor and system control unit locked in a physically secure enclosure that cannot be opened without major damage to its contents.

Why should I use a coprocessor if the microcontroller in the system control unit is fast enough to perform the SHA-1 computations?
(Same as with DS1963S token)

Why is the DS1963S purse file format not used with the DS1961S?
Due to its smaller scratchpad, writing a full 32-byte page requires four copy scratchpad operations and as many SHA-1 computations. The DS1961S also has no internal power source. For this reason, writing more data increases the risk of write cycles that occur at insufficient energy, which leads to data corruption or data that cannot be read without ambiguity. Originally, only the "purse A scheme" was used, which in a touch environment occasionally caused data failures when the contact between token and vending station broke just during a write cycle. As a remedy, the A-B Scheme was developed, which in any case keeps the new and the previous monetary value in the purse. This way, if the contact breaks, a mismatch in the length byte and CRC value will indicate that an update cycle did not complete. With this information the cycle can be finished at the next occasion and the data integrity is maintained. Extensive tests were made over the course of several days and well over 1 million debit cycles with noise injected in the 1-Wire line. With the Refresh Scratchpad command (see DS1961S data sheet) and the A-B Scheme, no loss of monetary value occurred during these tests.

Why is the SHA-1 input of the DS1961S designed as we know it?
The overriding objective was to be able to use a DS1963S as a coprocessor. For this to work, the DS1961S Copy Scratchpad command must be defined in such a way that the SHA-1 MAC can be computed by a DS1963S using a "Class B" SHA-1 subcommand - i.e. a subcommand in which the values of the counter and serial number fields are taken from the scratchpad. The four "Class B" subcommands are Sign Data Page, Validate Data Page, Compute First Secret, and Compute Next Secret. Of these, Sign Data Page is the only one, which does not set the HIDE flag. Therefore, the DS1961S Copy Scratchpad command must be defined so as to allow a DS1963S coprocessor using the Sign Data Page subcommand to calculate the MAC.
The DS1963S Sign Data Page subcommand, assuming that no Compute Challenge/Authenticate Host sequence has been completed, has M=0 and X=0. Therefore, the DS1961S Copy Scratchpad command must also have M=0 and X=0. The DS1961S Read Authenticated Page command must have M and X set to something different, so that a fraudulently obtained DS1963S cannot be used to manufacture a correct Read Authenticated Page MAC on-the-fly for fraudulent data. Therefore, it must have M=0 and X=1, so that it can be validated by a DS1963S Authenticate Host subcommand.
Therefore, the fundamental reason for the difference in the M and X settings for the Read Authenticated Page command between the DS1963S and the DS1961S is the requirement in the DS1961S for a SHA computation during the Copy Scratchpad command, which has no counterpart on the DS1963S. The differences, though counterintuitive, are legitimate and should not be changed.