Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2606981 C 2016/09/06

(11)(21) 2 606 981

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) Date de dépôt PCT/PCT Filing Date: 2006/05/05
(87) Date publication PCT/PCT Publication Date: 2006/11/09
(45) Date de délivrance/Issue Date: 2016/09/06
(85) Entrée phase nationale/National Entry: 2007/10/31
(86) N° demande PCT/PCT Application No.: CA 2006/000711
(87) N° publication PCT/PCT Publication No.: 2006/116871
(30) Priorité/Priority: 2005/05/05 (US60/677,816)

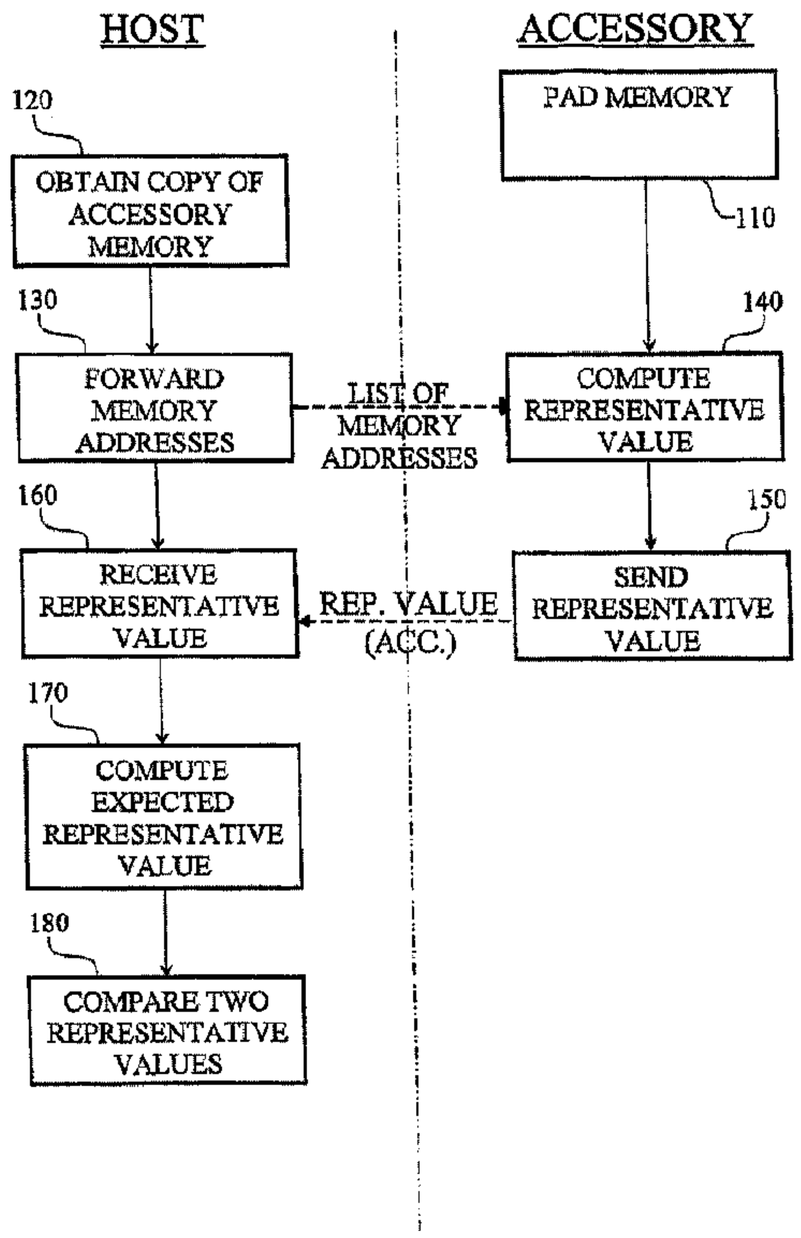
(51) Cl.Int./Int.Cl. *G06F 21/57* (2013.01), *G06F 12/00* (2006.01), *G06F 21/44* (2013.01)

(72) Inventeurs/Inventors:
VADEKAR, ASHOK, CA;
NEILL, BRIAN, CA

(73) Propriétaire/Owner:
CERTICOM CORPORATION, CA

(74) Agent: ROWAND LLP

(54) Titre : AUTHENTIFICATION DE RETROINSTALLATION SUR UN MICROLOGICIEL
(54) Title: RETROFITTING AUTHENTICATION ONTO FIRMWARE



(57) Abrégé/Abstract:
The present invention provides an inexpensive, software-based security-retrofit solution to verify the integrity of program code in embedded systems, or accessories, without resorting to expensive hardware changes. All unused memory on an accessory that could be used to store a program code image is filled with random data. A host system also locally stores a copy of the accessory's program image containing the random data. The host system sends the accessory a list of memory addresses or memory ranges on the accessory, which is always different and random in nature. The accessory will then produce a digest using values stored in the memory addresses as inputs to a secure hash function. The host system verifies the integrity of the embedded program code by verifying the resulting digest produced by and returned from the accessory.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 November 2006 (09.11.2006)

PCT

(10) International Publication Number
WO 2006/116871 A3

(51) International Patent Classification:
G06F 21/00 (2006.01) *G06F 12/00* (2006.01)

(21) International Application Number:
PCT/CA2006/000711

(22) International Filing Date: 5 May 2006 (05.05.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/677,816 5 May 2005 (05.05.2005) US

(71) Applicant *(for all designated States except US)*: **CERTICOM CORP.** [CA/CA]; 5520 Explorer Drive, 4th Floor, Mississauga, Ontario L4W 5L1 (CA).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **VADEKAR, Ashok** [CA/CA]; 250 Harris Street, RR #4, Rockwood, Ontario N0B 2K0 (CA). **NEILL, Brian** [CA/CA]; 2135 Donald Street, Burlington, Ontario L7M 3R3 (CA).

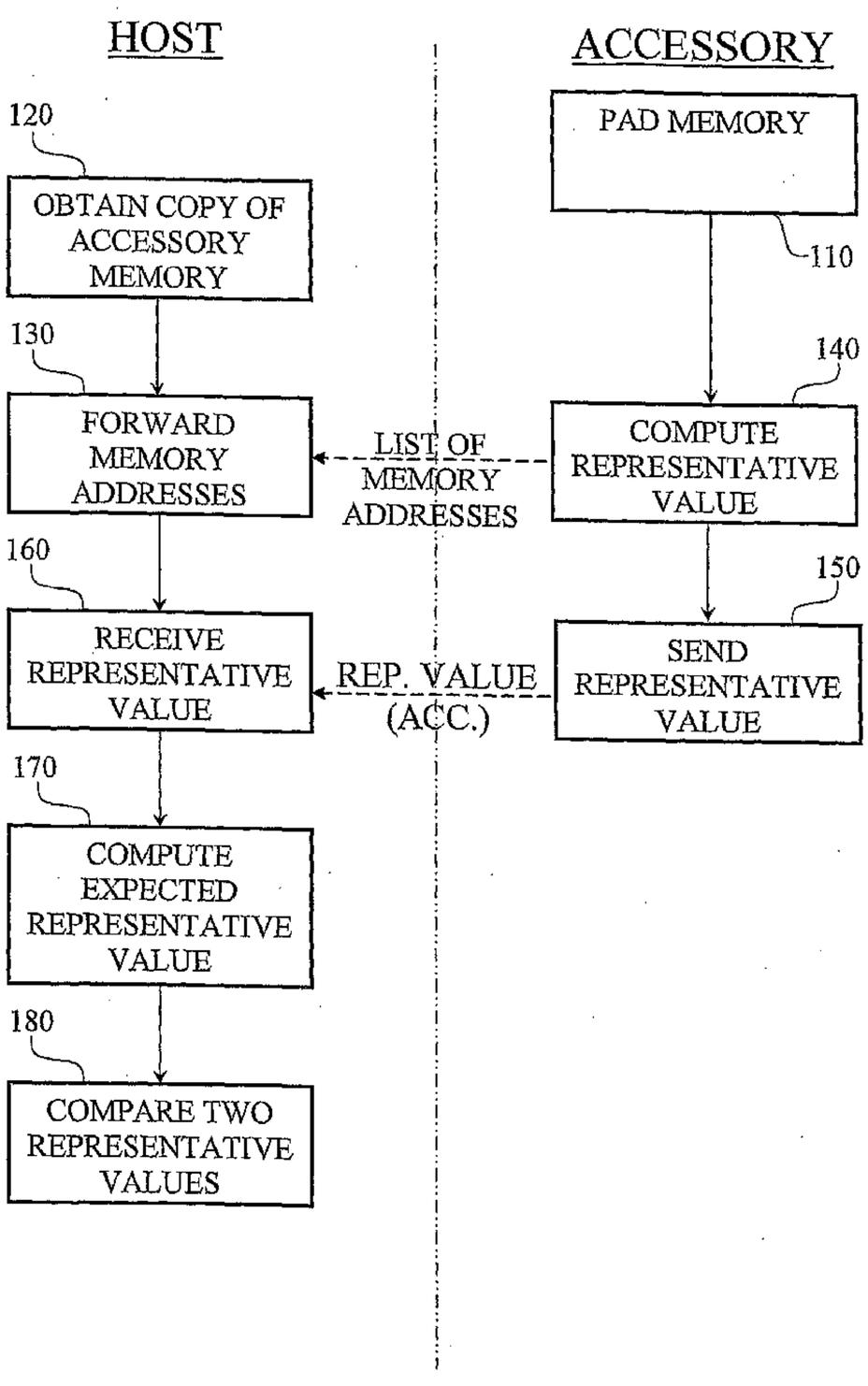
(74) Agents: **ORANGE, John, R., S.** et al.; Suite 2800, Commerce Court West, 199 Bay Street, Toronto, ON M5L 1A9 (CA).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: RETROFITTING AUTHENTICATION ONTO FIRMWARE



(57) Abstract: The present invention provides an inexpensive, software-based security-retrofit solution to verify the integrity of program code in embedded systems, or accessories, without resorting to expensive hardware changes. All unused memory on an accessory that could be used to store a program code image is filled with random data. A host system also locally stores a copy of the accessory's program image containing the random data. The host system sends the accessory a list of memory addresses or memory ranges on the accessory, which is always different and random in nature. The accessory will then produce a digest using values stored in the memory addresses as inputs to a secure hash function. The host system verifies the integrity of the embedded program code by verifying the resulting digest produced by and returned from the accessory.

WO 2006/116871 A3

WO 2006/116871 A3

Published:
— *with international search report*

(88) Date of publication of the international search report:
21 December 2006

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

RETROFITTING AUTHENTICATION ONTO FIRMWARE

FIELD OF INVENTION

[0001] The invention relates generally to the field of cryptography. In particular, the invention relates to providing an inexpensive, software-based security-retrofit solution to verify the integrity of program code in embedded systems.

BACKGROUND OF THE INVENTION

[0002] Computer systems typically use peripheral devices to supplement their functionality. Within this context, the computer system is called a host system and its peripheral devices are called accessories. Accessories are often devices capable of computation, as they are typically built using micro-processors or micro-controllers that can be programmed and re-programmed with program code, micro-code or firmware. The functionality and correct operation of these accessories are reliant on the correctness of the program code that resides in the accessory.

[0003] There are occasions when a fielded accessory needs to be upgraded or retrofitted because of a deficiency discovered in the program code. For example, it may be necessary to upgrade a high-risk instrument in high-value applications, such as medical devices. These types of devices are often regulated by the government. When a deficiency is discovered and re-design is underway, a quick stop-gap solution may be available. It may therefore be desirable that such a solution be applied to the devices already deployed in the field.

[0004] Individuals may maliciously alter the accessory's program code so that it performs unauthorized operations, from the perspective of the host system, yet fools the host system into believing nothing is amiss by reporting normal behavior. It is desirable that any unauthorized modification of the accessory's program code be detected before an upgrade is applied and that an upgrade is applied only if no such unauthorized modification is detected.

[0005] One solution to this problem is to design security into accessories before they are fielded, for example, by using secure micro-controllers that authenticate all micro-code upgrades before being accepted. However, it is sometimes the case that when an accessory is

first fielded, security concerns and risk levels are low, but over time, circumstances change such that the risk associated with the unsecured accessory unexpectedly increases. The cost to replace or make hardware changes to the accessory may be deemed too expensive, at which point, the host system has an unsecured accessory in a high-risk environment.

[0006] A naïve solution to this problem is to allow the host system to read all of the program code from the accessory, compute a digest using a secure hashing function, and compare the result to some locally maintained digest. There are two problems with this approach. First, the accessory could be reprogrammed such that it maintained a copy of the original program code, usurped read requests from the host system, and returned to the host system the original stored program code image. Second, the program code stored on the accessory may be too large to be effectively transmitted over a slow serial connection, or may not even be remotely accessible. In these instances, it is more effective to calculate a digest to be returned to the host system for validation.

[0007] Simply calculating a digest, even using a challenge-response mechanism, may not effectively defeat an altered accessory that is maintaining a copy of the original program code in memory somewhere on the accessory.

[0008] It is an object of the present invention to mitigate or obviate at least one of the above mentioned disadvantages.

SUMMARY OF THE INVENTION

[0009] In general terms, the invention comprises filling all unused memory on an accessory device, that could be used to store a program code image, with high entropy random data. A host system must also have access to a trusted copy of the accessory's memory image containing the random data.

[0010] The existing program on the accessory has an application program interface (API) that can be called by the host system. The accessory may be re-written where necessary to include an additional API, if the accessory does not already have one. The host system uses

the API to send the accessory a list of memory addresses or memory ranges on the accessory. This list is always different and truly unpredictable to prevent the list from being guessed by the accessory ahead of time. The accessory will then produce a representative value that is determined from the list and the values of the accessory device's memory referenced by the list. Preferably, the representative value is a digest using memory values at the supplied memory addresses as inputs to a secure hash function. The resulting digest, produced by the accessory, is returned to the host system for validation.

[0011] By padding memory with random data, an attacker is prevented from having enough room on the accessory to replace the program image and maintain the old program image. The accessory cannot be altered by an attacker by replacing the program image alone without adding additional memory to the accessory first, requiring the attacker to make a hardware change. By padding the memory with random data and thus implicitly including the random data in the authentication process, an attacker is further prevented from being able to make unauthorized changes to the program image without being detected.

[0012] The end result is that the host system can have some assurance that the accessory's program code has not been maliciously changed.

BRIEF DESCRIPTION OF DRAWINGS

[0013] For the purposes of description, but not of limitation, an embodiment is explained in greater detail by way of example with reference to the accompanying drawings, in which:

[0014] Figure **1** is a block diagram showing a host system communicating with an accessory;

[0015] Figure **2** is a flowchart diagram illustrating steps of a method for verifying the integrity of program code in the accessory shown in Figure **1**; and

[0016] Figure **3** shows schematically unused memory space on the accessory shown in Figure **1** padded with random data and memory ranges on the accessory randomly selected by the host system.

DETAILED DESCRIPTION OF EMBODIMENTS

[0017] The description which follows, and the embodiments described therein, are provided by way of illustration of an example, or examples, of particular embodiments of the principles of the present invention. These examples are provided for the purposes of explanation, and not limitation, of those principles and of the invention. In the description which follows, like parts are marked throughout the specification and the drawings with the same respective reference numerals.

[0018] Referring to Figure **1**, there is shown a host **20** communicating with an accessory **22**. Host **20** is generally a computer system that has a CPU **24**, host memory device **26** accessible to CPU **24**, host storage media **28**, also accessible to CPU **24**, and some input and output devices (not shown). Application program **30** executes on CPU **24**. Application program **30** may be stored on host storage media **28**, which may be permanently installed in host **20**, removable from host **20** or remotely accessible to host **20**. Application program **30** communicates with and directs the operation of accessory **22**.

[0019] Accessory **22** generally has some computation power. Typically, it has a microprocessor or a microcontroller **32**. Accessory **22** may also have other types of programmable processors that have adequate computation power. For example, accessory **22** may be one equipped with a DSP (digital signal processor) or a FPGA (Field-Programmable Gate Array). The processor, or microcontroller **32** generally has access to a memory storage space that may be divided into accessory memory device **34** and volatile memory device **36.** Firmware program code **38**, or embedded program code or micro-code, executes on microcontroller **32** and supplements application program **30** executing on host **20** to further control operations of accessory **22**.

[0020] Firmware program code **38** and persistent (constant) data may be stored on accessory memory device **34**, or any persistent medium. Volatile data, i.e., data related to operative state of the program, may be stored on volatile memory device **36**. Either accessory memory device **34** or volatile memory device **36** may also be used to store semi-persistent

data, i.e., data persistent across power cycles but that may be programmatically modified during each power cycle.

[0021] Data link **40** provides a communication channel between application program **30** and firmware program code **38** when needed. Data link **40** may be wired or wireless. For example, it may be a connection cable or a radio frequency connection. It may be a direct connection between host **20** and accessory **22** or relayed through some intermediary host systems. The data link **40** may be permanent, or more preferably, a connection that is established on demand.

[0022] Each of application program **30** and firmware program code **38** has application program interfaces (APIs) for communicating with each other. In particular, as will be described later, firmware program code **38** has an API that application program **30** calls to forward a list of memory addresses and an API through which firmware program code **38** returns a representative value such as a digest calculated based on the contents of a list of memory addresses. Although conceptually, firmware program code **38** is described to have two separate APIs, these two APIs may be implemented as a single API in practice. Application program **30** also has corresponding APIs for sending the list of memory addresses and for receiving the returned representative value.

[0023] While the distinction is made here that there is a host memory device **26** which tends to be used for storing more volatile data and a host storage media **28** which tends to be used for storing more persistent data, host **20** may have only a single data storage device for storing both volatile and persistent data. Similarly, accessory **22** may have only a single memory device for storing both volatile and persistent data.

[0024] As will be appreciated, host **20** may be a general purpose computer, a custom tailored special purpose computer, or some other programmable computation devices. Further, although host **20** is shown and described as a single computer system, it is only for the convenience of description. Host **20** is understood to be collectively the combined system, which may include several computer systems, for performing the tasks as described. As will be appreciated by one of skilled in the art, the computation and storage performed by

host **20** may be distributed among several networked computers without affecting the functioning of the system and the performance of the method described herein.

[0025] Referring to Figures **2** and **3**, an example of a method of verifying the integrity of firmware program code **38** is described in detail. In Figure **2**, to the left of the dotted line are operations generally performed by host **20** or on components of host **20**. To the right of the dotted line in Figure **2** are operations generally performed by accessory **22** or on components of accessory **22**.

[0026] Typically, the size of firmware program code **38** is smaller than the size of memory available on accessory memory device **34.** In other words, accessory memory device **34** typically has unused memory space, i.e., space not occupied by firmware program code **38** nor data associated with firmware program code **38**. To limit the ability of an adversary to use the unused memory space for storing any unauthorized code, prior to verification, all unused memory space is filled with random data.. The random data preferably has high entropy. High entropy random data tends to be difficult to compress while data that possess randomness but of low entropy tends to compress well. Preferably, all memory space on accessory memory device **34** is occupied by incompressible data. Filling all memory space with incompressible data discourages an adversary from compressing the data and thereby gaining memory space. A number of random number generation algorithms are available for generating high entropy random data. For example, the random number generation algorithms specified in "Change Notice 1" of Federal Information Processing Standards Publication 186-2, "Digital Signature Standard", issued by the National Institute of Standards and Technology (NIST) of the United States of America, may be used. In addition, block encrypting data of lower entropy tends to produce output of higher entropy.

[0027] At step **110**, all unused memory on accessory device **22** is filled with random data and a trusted copy of the padded memory image is retrieved for later reference or use. Although this step may be performed by accessory device **22**, there are existing tools that allow one to fill all unused memory with randomly generated high entropy data. Generally, the randomness of the data is controlled by the tool that is employed to fill the unused

memory space. Preferably, memory padding is performed at a development facility or manufacturing site before the accessory is released to the market.

[0028] After padding with random data, all memory space on accessory **22** is occupied. Referring to Figure **3**, accessory memory space **200** may be divided into contiguous program code and persistent data space **202** and unused memory space **204**. Unused memory space **204** is filled with random data **206**. Although Figure **3** shows that firmware program code **38** occupies a contiguous segment of memory, namely a contiguous portion of program code and persistent data space **202**, it is possible that firmware program code **38** may occupy several disjoint memory segments. Similarly, persistent (constant) data may occupy a contiguous segment of memory or several disjoint memory segments. Whether or not program code and persistent data space **202** is contiguous, all the unused memory space will be filled, or padded with random data.

[0029] As will be appreciated, accessory memory space **200** may include memory space occupied by volatile data on volatile memory device **36**. Memory elements that do not change over the validation process (or change in a well defined fashion) can be validated. Accessory memory space **200** may also include memory space occupied by data on a "peripheral" memory device, such as a serial EEPROM. In fact, accessory memory space **200** may even include portions outside the physical space of both accessory memory device **34** and volatile memory device **36** (but these addresses fold back as shadow of physical addresses). These, of course, require that accessory **22** (and host **20**) support a memory addressing scheme that can address any and all available memory spaces and support special memory configurations such as bank-switching, overlays and shadowing. In fact, different memory addressing schemes and special memory configurations may be exploited to further enhance the security. For example, when validated memory addresses correspond to memory spaces outside the physical memory space, it would be difficult for a modified program requiring a compatible but larger memory footprint to emulate the validation process of the original program and the smaller device while running on the necessarily larger device.

[0030] An image of memory on accessory **22,** with no unoccupied memory spaces left, is retrieved for later reference, or use. The image is generated and saved in a trusted fashion. For example, the image may be obtained from accessory **22** during a trusted operation, such as programming at a manufacturing site of accessory **22**. It is also possible that the factory programs accessory images in bulk and the image is generated elsewhere at a development facility that also generates host code and data images. However the image is generated, the image is produced and provided in a trusted fashion. When the image is saved, the image is also saved in a trusted fashion. The end result is that a trusted copy of the image can be obtained when needed.

[0031] Host **20** obtains a trusted copy of the image of memory on accessory **22,** with no unoccupied memory spaces left, at step **120**. In general, host **20** does not store the trusted image locally in order to minimize the risk of the image being tempered with. Instead, host **20** is provided with access to a trusted copy of the image. Although only one trusted memory image is referenced here, it is possible that several different memory images of accessory **22** must be made available to host **20**. This is because accessory **22** may have been upgraded several times over time by its manufacturer. Each previous upgrade would result in a different memory image. When accessory **22** is upgraded in the field, accessory **22** may correspond to any one of the several upgrades, or even its original version.

[0032] As will be appreciated, although this step is described as the next following the saving of a trusted copy of the memory image, these two steps may be many days, months or even years apart. It is possible that the trusted copy of the memory image is saved during manufacturing of the accessory and the retrieval of a trusted copy of the image happens many years later when a fix is applied to the accessory deployed in the field. Further, although this step is described as a first step on the host side, this is not necessary. This step needs to be completed prior to computing the expected representative value and may be performed any time before the computation.

[0033] Referring to Figure **2**, host **20** at step **130** sends accessory **22** a list of memory addresses to initiate the verification process. The list may be a range of memory addresses or

several ranges of memory addresses on the accessory memory device **34**. For example, Figure **3** shows three randomly selected memory segments, or memory ranges: a first memory range **208**, or memory segment **A**, between first starting address **210** and first ending address **212**, a second memory range **214**, or memory segment **B**, between second starting address **216** and second ending address **218**, and a third memory range **220**, or memory segment **C**, between third starting address **222** and third ending address **224.**

[0034] The list, or the ranges of memory addresses, is random in nature or at least unpredictable so that the ranges cannot be anticipated by accessory **22**, i.e., the list cannot be anticipated by an adversary attempting to alter maliciously the firmware code stored on accessory **22**. Not only the selection of starting and ending points is unpredictable, so is the ordering of the selected memory ranges. For example, the list may contain the addresses of segment C as the first range, the addresses of segment A as the second range and the addresses of segment B as the third range. Changing the ordering of the selected memory ranges produces a different list and would also produce a different representative value at step **140**, as will be described below. The list is typically generated by host **20**. However, the list may be generated in any way as long as the list is truly unpredictable and different each time it is generated. As host **20** generally has more computation power, host **20** typically generates and sends the list to accessory **22**.

[0035] In one exemplary implementation, one of the memory address ranges included in the list always contains the program code that is considered critical to the operation of accessory **22**. In other words, host **20** always includes a memory segment that contains the entire critical code of application program **30**. Although the memory containing the critical code is always included in the selected memory ranges, the memory segment containing the critical code may be randomly arranged within the list, as described above.

[0036] Upon receiving the list of memory addresses, accessory **22** produces at step **140** a value that takes as inputs the list of memory addresses and the memory values at the supplied addresses. The value generated is representative of the memory values at the supplied addresses. A number of algorithms may be used to produce the value, provided the value

produced is representative of the list and the actual values at the supplied memory addresses. For example, the values of the memory at the supplied memory addresses may be first read and then concatenated together into a string. The string will then be the representative value. Preferably, the representative value is a digest computed using a secure hashing algorithm. The secure hashing algorithm computes the digest using the memory values at the supplied memory addresses as its input. Any digesting algorithms with the security properties of the SHA (Secure Hash Algorithm) may be used. The secure hash function may be one based on SHA-1 or MD5, for example. Preferably, one can use a digesting algorithm with the security properties of the SHA already existed in an accessory implementation; otherwise, one of SHA-1 or SHA-256, depending on the efficiency on the specific processor type of accessory **22**, may be used. Accessory **22** returns, i.e., sends, the resulting representative value to host **20** at step **150**, which is received by host **20** at step **160**.

[0037] The example shown in Figure **3** has first memory range **208** corresponding to memory spaces occupied entirely by persistent data and firmware program code **38**, third memory range **220** corresponding to unused memory spaces **204** filled with random data, and second memory range **214** corresponding to a section of memory that contains partly firmware program code **38** and partly random data. As will be appreciated, other selections are possible. What is important is that the possible set of lists (that can be requested by host **20**) be sufficiently large and that the selection of addresses is truly unpredictable. This tends to discourage precomputing all (or even a significant subset of) corresponding representative values as the computation may be expensive or even infeasible, both from a storage and computation perspective.

[0038] Firmware program code **38** may have modules implementing one or several of these secure hashing algorithms. Where only one secure hashing algorithm is implemented, the accessory computes a digest using the implemented secure hash function, taking the memory ranges received as input, and sends the resulting digest to the host system. Where the firmware program code **38** implements more than one secure hashing algorithm, a digest using one of the implemented secure hashing algorithms is produced. The resulting digest,

together with an indication of the secure hashing algorithm used, is sent to host **20**. As will be appreciated, either host **20** or accessory **22** may select a particular secure hashing algorithm and inform the other the secure hashing algorithm used to produce the digest.

[0039] When retrofitting a firmware, the firmware program code **38** may not have an API for receiving the list of randomly generated memory addresses. It also may not have a module for producing a digest using a secure hash algorithm. In order to prepare such an accessory for a secure upgrade, namely, an upgrade that is performed only if the firmware code can be authenticated, it will be necessary to first retrofit authentication function on to the accessory. In other words, it will be necessary to first rewrite the existing program on accessory **22** to include an additional API that can be called by host **20** for receiving the list of memory addresses and an additional API that can return to the host a representative value computed from the list. The rewritten firmware program will also include a module for computing a representative value or a module or modules for implementing secure hashing algorithms for computing a digest.

[0040] After host **20** receives the resulting digest or the representative value at step **160** from accessory **22**, host **20** uses the trusted copy of the memory image of the accessory to compute an expected representative value at step **170**. Preferably, the same algorithm used by accessory **22** is used by host **20** to produce the expected representative value. But this is not necessary. The algorithm used by host **20** only needs to be equivalent to that used by accessory **22** so that the expected representative value will be the same as the received representative value. If accessory **22** implements more than one secure hashing algorithm but uses only one to compute a digest, host **20** selects the same or equivalent algorithm to compute the expected representative value.

[0041] At step **180**, host **20** compares the representative value received from accessory **22** with the expected representative value that was computed locally to verify the accessory program image. Firmware program code **38** will not be authenticated if these two values are not identical or equivalent to each other.

[0042] As described, the first step of this method is to ensure that all memory on the accessory is occupied. If the compiled program that is used to implement the accessory's functionality is smaller than the physical memory on the accessory, the remainder is padded with truly random and high entropy data. The padded data becomes part of the memory image that is made available to the host system during a retrofitting upgrade.

[0043] In a further exemplary implementation, host 20 sends a string of data at step 130 along with the list of memory addresses to accessory 22. The string may be random, or it may include an identification information such as the unique identity of the accessory (or even the host system). When the string is used as auxiliary input to the secure hash function, the method is typically referred to as a challenge-response method. However, it still relies on random padding data in the memory image to prevent the injection of illicit program code into the accessory.

[0044] Various examples have now been described in detail. Those skilled in the art will appreciate that numerous modifications, adaptations and variations may be made to the examples without departing from the scope of the invention. The invention is to be limited only by the scope of the appended claims.

CLAIMS

What is claimed is:

1. A method of verifying integrity of program code embedded in an accessory, said accessory having a memory device for storing the program code and data associated with the program code, all unused memory space of the memory device being filled with random data, a host being in operable communication with a memory image of the memory device, said method comprising the steps of the host:

sending a list of start and end points defining memory address ranges to the accessory, wherein said start and end points are unpredictable to the accessory, and wherein said list specifies said memory address ranges in a sequence unpredictable to the accessory;

receiving a value from the accessory, said value being generated from and representative of values of memory at memory addresses in said memory address ranges arranged according to said sequence specified in said list;

producing an expected value from the memory image and said list; and,

comparing said value received from the accessory with said expected value;

wherein the integrity of the program code is verified if said received value is equivalent to said expected value.

2. The method of claim 1, wherein said value is a concatenation of said values of memory at said memory addresses in said memory address ranges arranged according to said sequence specified in said list.

3. The method of claim 1, wherein said value is a digest computed using a secure hashing algorithm with said values of memory at said memory addresses as input thereto and said expected value is an expected digest computed using a second secure hashing algorithm equivalent to said secure hashing algorithm.

4. The method of claim 3, further comprising the step of:

sending a random data string to the accessory, wherein the digest is computed taking the random data string received as an additional input, and the expected digest is produced taking the random data string as a second input.

5. A method of preparing an accessory for authenticating its firmware program code, said firmware program code and data associated with said firmware program code being stored in a memory device of said accessory, said method comprising the steps of:

providing said firmware program code with a first API for receiving a list of start and end points defining memory address ranges from a host, addresses in said memory address ranges corresponding to memory addresses addressable by said firmware program code, and wherein said start and end points are unpredictable to the accessory, and wherein said list specifies said memory address ranges in a sequence unpredictable to the accessory;

providing said firmware program code with a second API;

providing said firmware program code with a cryptographic program for producing a representative value from said list and values of memory at said addresses in said memory address ranges according to said sequence specified in said list, said second API returning said representative value to said host; and,

filling all memory space of memory device unoccupied by said firmware program code and said data with random data, and providing a memory image of said memory device for reference;

wherein said firmware program code is authenticated if in an authentication operation, said representative value is equivalent to an expected representative value, said expected representative value being computed by said host from a copy of said memory image using said list.

6. The method of claim 5, wherein said representative value is a digest computed using a secure hashing program.

7. The method of claim 6, wherein said expected representative value is an expected digest computed using a second secure hashing algorithm equivalent to said secure hashing algorithm.

8. The method of claim 7, further comprising the step of:

sending a random data string to the accessory, wherein the digest is computed taking the random data string received as an additional input, and the expected digest is produced taking the random data string as a second input.

9. The method of any one of claims 1 to 8, wherein at least one of said address ranges is selected to include a portion of the program code comprising the entire critical code of the program code.

10. An accessory having an authenticatable firmware program code, said firmware program code and data associated with said firmware program code being stored in a memory device of said accessory, said accessory having an externally stored memory image of said memory device, said firmware program code comprising:

an interface for receiving an input from a host and for returning a value to the host, said input comprising start and end points defining ranges of memory addresses selected by the host, wherein said start and end points are unpredictable to the accessory; and,

a program for producing the value, the value being a function of said input and values of memory at memory addresses in said defined ranges on said memory device;

wherein said host produces an expected value from a trusted copy of said externally stored memory image using said input and compares said expected value with said value returned from said firmware program code , and wherein said firmware program code is authenticated if said value returned from said firmware program code is equivalent to said expected value.

11. The accessory of claim 10, wherein said program for producing the value is a secure hashing program and said value is a digest produced by said secure hashing program.

12. The accessory of claim 10, wherein said input is a random value generated by the host.

13. The accessory of claim 10, wherein said interface comprises:

an API for receiving a list of said ranges of memory addresses from said host, addresses in said ranges of memory addresses corresponding to memory addresses on said memory device

addressable by said firmware program code and sequences of said address ranges in said list being unpredictable to the accessory; and

a second API for returning said value to said host.

14. The accessory of claim 10, wherein all memory space of said memory device not occupied by said firmware program code and said data is entirely filled with random data.

15. The accessory of claim 13, wherein said sequence is randomized.

16. The accessory of any one of claims 10 to 15, wherein at least one of said address ranges is selected to include a portion of the firmware program code comprising the entire critical code of the firmware program code.

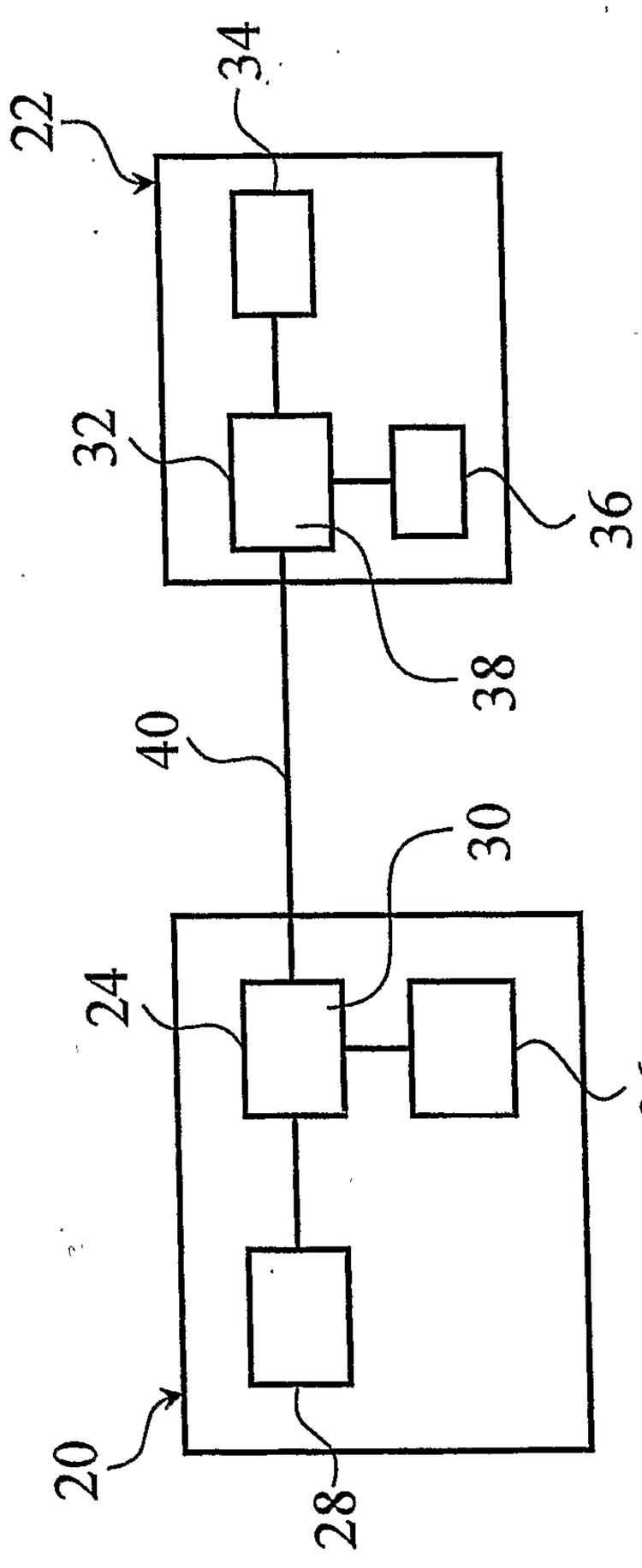


FIG. 1

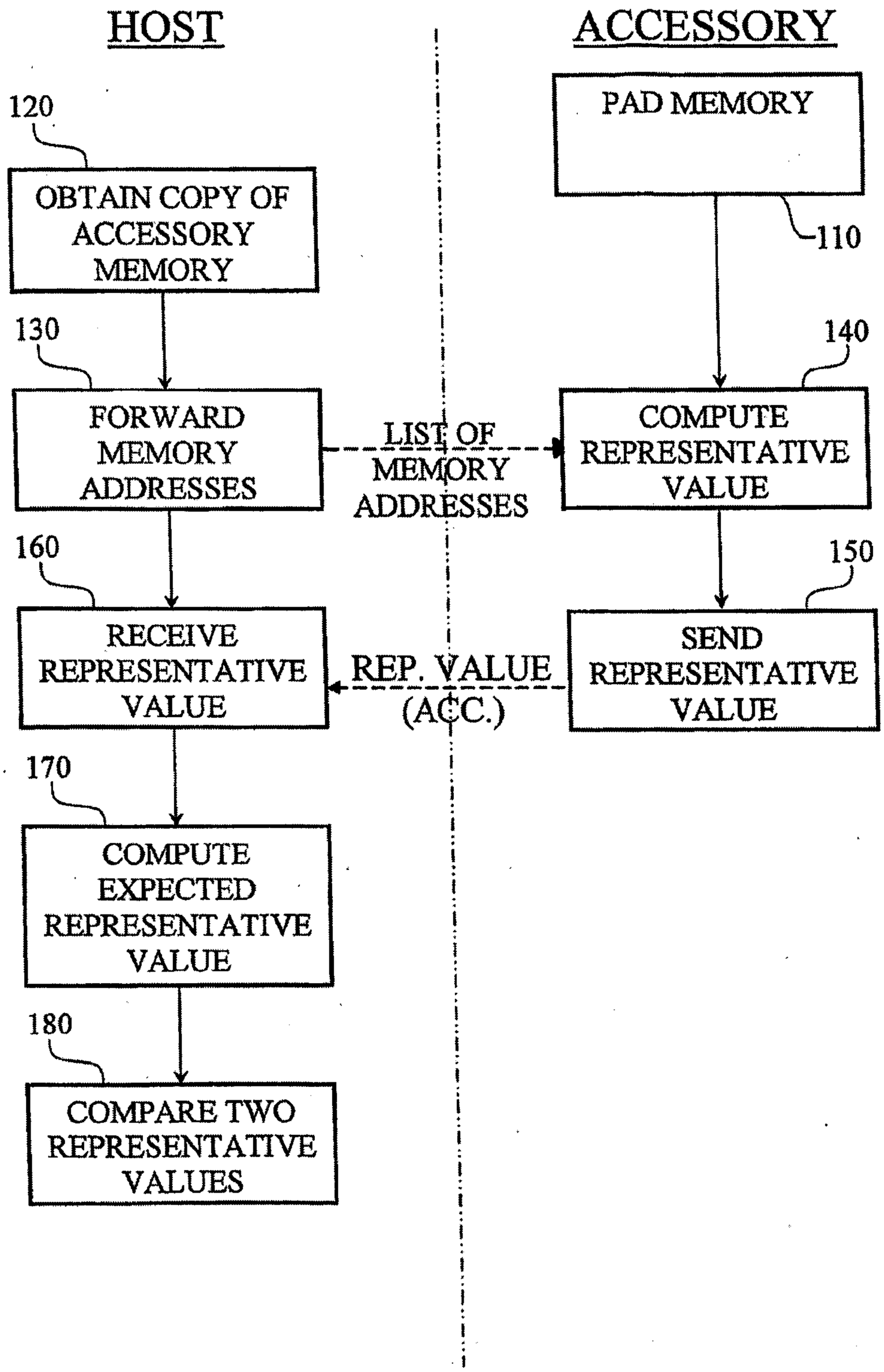

HOST
ACCESSORY
120
OBTAIN COPY OF ACCESSORY MEMORY
PAD MEMORY
110
130
FORWARD MEMORY ADDRESSES
LIST OF MEMORY ADDRESSES
140
COMPUTE REPRESENTATIVE VALUE
160
RECEIVE REPRESENTATIVE VALUE
REP. VALUE (ACC.)
150
SEND REPRESENTATIVE VALUE
170
COMPUTE EXPECTED REPRESENTATIVE VALUE
180
COMPARE TWO REPRESENTATIVE VALUES

FIG. 2

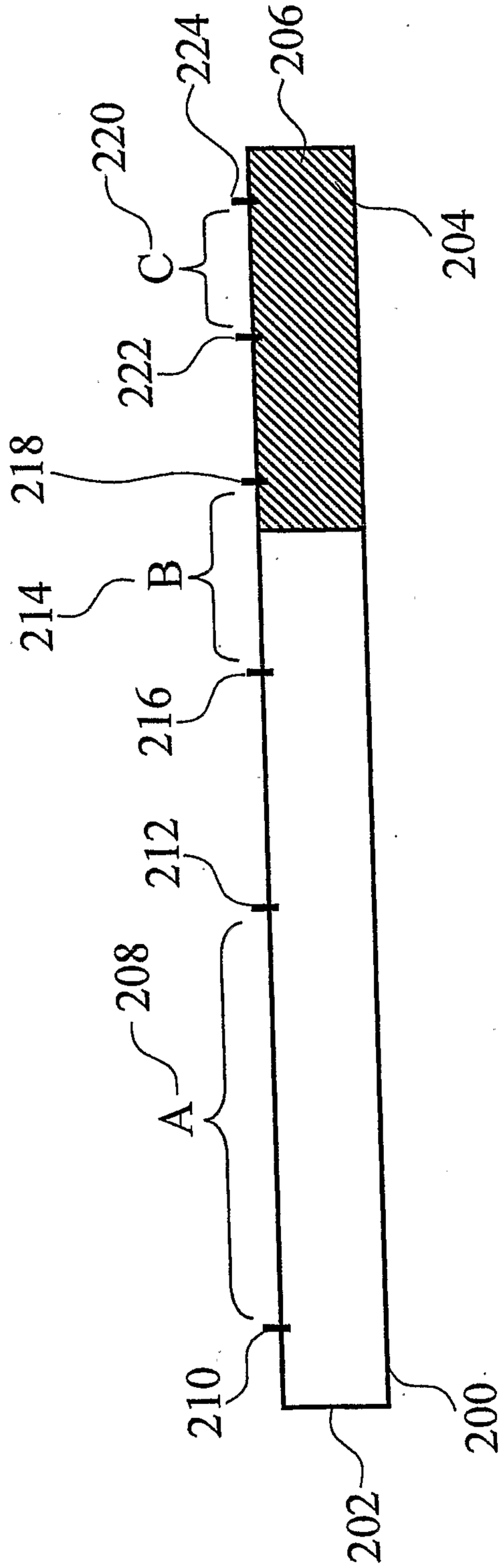
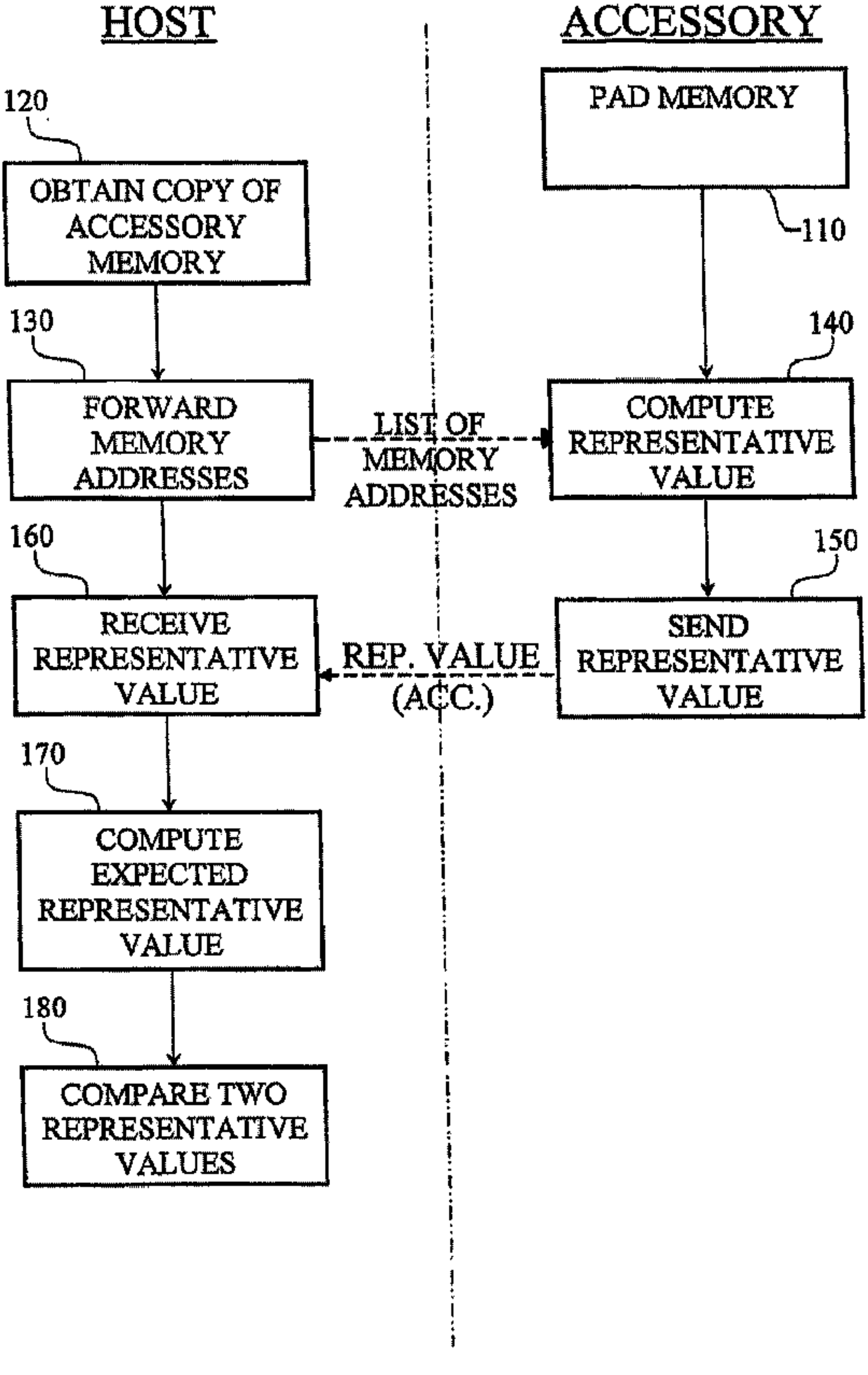


FIG. 3


HOST
ACCESSORY
120
OBTAIN COPY OF ACCESSORY MEMORY
PAD MEMORY
110
130
140
FORWARD MEMORY ADDRESSES
LIST OF MEMORY ADDRESSES
COMPUTE REPRESENTATIVE VALUE
160
150
RECEIVE REPRESENTATIVE VALUE
REP. VALUE (ACC.)
SEND REPRESENTATIVE VALUE
170
COMPUTE EXPECTED REPRESENTATIVE VALUE
180
COMPARE TWO REPRESENTATIVE VALUES