

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
13 April 2006 (13.04.2006)

PCT

(10) International Publication Number
WO 2006/038103 A1

(51) International Patent Classification:
G07F 7/10 (2006.01)

(21) International Application Number:
PCT/IB2005/002975

(22) International Filing Date: 6 October 2005 (06.10.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/617,539 9 October 2004 (09.10.2004) US
11/243,282 4 October 2005 (04.10.2005) US

(71) Applicant (for all designated States except US): **AXALTO S.A** [FR/FR]; 50, avenue Jean Jaurès, F-92120 Montrouge (FR).

(72) Inventor: **PREVOST, Sylvain**; c/o Axalto SA, Intellectual Property Dpt, 50 avenue Jean Jaures, F-92120 Montrouge (FR).

(74) Agent: **JACQUOT, Ludovic**; C/O Axalto S.A., 50, avenue Jean Jaurès, F-92120 Montrouge (FR).

(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, 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).

Declaration under Rule 4.17:

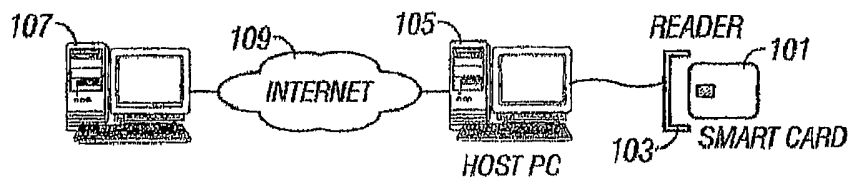
— of inventorship (Rule 4.17(iv))

Published:

— with international search report

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.

(54) Title: SYSTEM AND METHOD FOR POST-ISSUANCE CODE UPDATE EMPLOYING EMBEDDED NATIVE CODE.



(57) Abstract: Updating system software of a resource-constrained device having a microprocessor. The system software is updated by embedding native code instructions in an application program that is downloaded onto the resource-constrained device. The native code instructions are selected from the instruction set requiring no processing prior to execution by the microprocessor. In response to detecting that an application contains embedded native code, passing the embedded native code directly to the microprocessor for execution.



WO 2006/038103 A1

**SYSTEM AND METHOD FOR POST-ISSUANCE CODE UPDATE
EMPLOYING EMBEDDED NATIVE CODE**

5 **[01] BACKGROUND OF THE INVENTION**

[02] 1.0 Cross-Reference to Related Applications

 [03] This invention claims priority pursuant to 35 U.S.C. 119 of U.S. Provisional Patent Application Serial No. 60/617,539, filed on October 9, 2004. This Provisional Application is hereby incorporated by reference in
10 its entirety.

[04] 2.0 Field of the Invention

 [05] The present invention relates generally to updating system software loaded onto smart cards and more particularly to a mechanism for updating system code for smart cards with native code during the
15 smart card life cycle.

[06] DESCRIPTION OF THE RELATED ART

 [07] Smart cards are small personal computing devices that are used to protect very sensitive information. Smart cards may be used to perform banking functions, provide access to health records, personalization of
20 computer network access, secure building access, and many more functions. Smart cards are also used as subscriber identity modules (SIM) in certain mobile telephony networks.

 [08] A crucial selling point of smart cards is the security of the data stored thereon or accessed through the use of smart cards.

[09] A recent trend in smart card technology is so called multi-application smart cards. These cards may be programmed with multiple disjointed application programs. For example, the same card may be used to access both banking records as well as provide health care information.

5 Examples of such cards include the Cyberflex family of cards from Axalto Inc.

[10] The application programs are usually written in a high level language that is compiled off card and loaded onto the card in an in the form of a program written in a non-native instruction set (e.g., Java
10 bytecodes, IL-opcodes). The system software on the card transforms the non-native code into instructions in a smart card chip native instruction set either through interpretation or just-in-time compilation for execution.

[11] For security reasons, to prevent applications to access data or resources that are off-limits to application programs, many primitives that
15 are available to card system software are not made available to application programs.

[12] A common feature of multi-application smart cards is that the application programs may be loaded onto the smart card after the card has been issued by the manufacturer or even after an end-user has taken
20 possession of the card. Each such application program in a multi-application smart card is stored in some form of programmable memory on the smart card. Such post-manufacture programmability of smart cards provides increased flexibility and power of use of the smart cards.

[13] Thus, after a card has been issued, it is possible to update the functionality of the card by downloading new or updated application programs onto the card. However, the post-issuance update of the system software is very problematic because of the security risks involved in replacing system modules. Usually, updating of the system functions is de-activated when the card has been made ready for issuance.

[14] This de-activation of system updates causes a number of problems. For one, it prevents new functionality to be added to a card that in theory is available in that portion of the native code that is made off-limits to application programs. Some functions that an operator may wish to add to a card may require access to the chip instruction set for performance reasons. Second, having the system fixed prevents the update of the system to correct bugs or other defects.

[15] Furthermore, while it is possible to add functionality to the smart card post-issuance by loading new applications onto the card, because of the restrictions on what low level primitives such applications may utilize, it is not practical to use that mechanism for system updates.

[16] In the prior art, in order to update the system, it was necessary to recall the smart card which needed the system update. With cards deployed in large quantities, recalling all cards in a deployment is impractical. Alternatively, entirely new cards may be manufactured for the purpose of replacing the cards that do not have the desired functionality. However, besides cost issues, that is undesirable because

these new cards would not have all the applications that may have been loaded onto a smart card after issuance.

[17] Accordingly, from the foregoing it is apparent that there is a hitherto unresolved need for a system and method for permitting the
5 secure update of the system software for a smart card post-issuance while allowing access to native code instructions the use of which are not available to application programs.

[18] BRIEF DESCRIPTION OF THE DRAWINGS

[19] Figure 1 is a schematic illustration of the operating environment in which a smart card according to the invention may be used to provide secure computing services.

5 [20] Figure 2 is a schematic illustration of an exemplary architecture of a resource-constrained device.

[21] Figure 3 is a schematic illustration of a software architecture for a resource-constrained device.

[22] Figure 4 is a schematic illustration of a software architecture
10 according to the invention in which one application program as illustrated in Figure 3 in which system functionality is updated with application programs with embedded native code according to the invention.

[23] Figure 5 is a flow chart illustrating the operations of the loader of Figure 4 according to the invention.

[24] DETAILED DESCRIPTION OF THE INVENTION

[25] In the following detailed description and in the several figures of the drawings, like elements are identified with like reference numerals.

[26] As shown in the drawings for purposes of illustration, the invention
5 is embodied in a system and method for providing secure post-issuance updates to system functions. The invention accomplished the aforesaid advance in smart card technology by providing a mechanism in which application programs may have microprocessor native code embedded therein, which the on-card virtual machine can recognize as such and
10 which the virtual machine passes directly to the central processing unit of the smart card for processing.

[27] Figure 1 is a schematic illustration of the operating environment in which a resource-constrained device according to the invention may be used to provide secure communication with a remote entity. A resource-
15 constrained device 101, for example, a smart card, is connected to a computer network 109, for example, the Internet. The resource-constrained device 101 may be connected to the computer network 109 via a personal computer 105 that has attached thereto a card reader 103 for accepting a smart card. However, the resource-constrained device 101
20 may be connected in a myriad of other ways to the computer network 104, for example, via wireless communication networks, smart card hubs, or directly to the computer network 109. The remote node 105 is a computer system of some sort capable to implement some functionality that may

either seek access to information on the smart card 101 or to which the smart card user may seek access. For example, the remote node 107 may be executing banking software that a user of the smart card 101 is seeking to obtain access to. The smart card 101 may then provide some access
5 control functionality or may even be an electronic purse to which funds are downloaded from the remote computer.

[28] The scenario of Figure 1 is presented here merely for the purpose of providing an example and must not be taken to limit the scope of the invention whatsoever. Only the imagination of designers limits the
10 myriad of possible deployment scenarios and uses for smart cards.

[29] Figure 2 is a schematic illustration of an exemplary architecture of a resource-constrained device 101. The resource-constrained device 101, e.g., a smart card has a central processing unit 203, a read-only memory (ROM) 205, a random access memory (RAM) 207, a non-volatile memory
15 (NVM) 209, and a communications interface 211 for receiving input and placing output to a device, e.g., the card reader 102, to which the resource-constrained device 101 is connected. These various components are connected to one another, for example, by bus 213. In one embodiment of the invention, the SSL/TLS module 103, as well as other software modules
20 shown in Figure 1, would be stored on the resource-constrained device 101 in the ROM 206. During operation, the CPU 203 operates according to instructions in the various software modules stored in the ROM 205.

[30] Figure 3 is a block diagram of an exemplary software architecture 300 that one may find implemented on a smart card 101. The software

architecture 300 includes several application programs 301. These are loaded onto the smart card by a loader 303. The application programs 301 would typically be loaded into the non-volatile memory 209. However, in other scenarios an application program may be permanently written onto the smart card at manufacture by having it stored in the ROM 205. If the smart card 101 were called upon to execute a program for only one session, it would be possible to have the program loaded in the RAM 207. However, that would be a rare circumstance. On the other hand, during execution of an application program, it is indeed possible that certain portions of the application program are loaded into the RAM 207.

[31] In this example, a several application programs 301 are executed by the CPU 203 under the control of instructions of an interpreter 305. The interpreter 303 may, for example, be a Javacard Virtual Machine as found on the Cyberflex smart card family from Axalto Inc. of Austin, Texas. In alternative embodiments, the application programs 301 are compiled into executable code and do not require further interpretation by the interpreter 305. However, in such embodiments, the job control would be managed by some operating system program that would take the place of the interpreter 303.

[32] The interpreter 303 is usually a static component of a smart card 101 and would therefore be loaded into the ROM 205. The interpreter 303 may also be burned into some form of firmware. In another alternative the interpreter 303 may be stored in the non-volatile memory 209.

[33] In most embodiments of the invention, the smart card software architecture 300 also includes some system functions 307. System functions 307 may include security functionality, cryptography functionality, and utility libraries that may be called by application programs 301.

[34] The application programs 301 may access functions provided by the smart card system software 307 by issuing calls through an application program interface 309.

[35] Figure 4 is an alternative software architecture for a smart card in which native code bearing applications are employed to update system functionality according to the invention. In this architecture the software for the smart card are divided into application programs 401 and system functions 403. Generally, application programs may be loaded onto the smart card at any time during the life cycle, e.g., during issuance or even during the usage phase. The system functions 403 contain programs that control system functionality of the smart card. For example, the system functions include a loader 405 that operates to load new application programs onto the smart card, a file system 407 for managing data files stored on the smart card, and a virtual machine 409 for executing the application programs 403. Of course, the various application programs 401 and system programs 403 are interconnected, e.g., so that an application program 401 may call upon the file system 407 to access data and so on, however, those connections are not shown in the Figure 4.

[36] In a preferred embodiment of the invention, the system functions 403 may be updated with new functionality by downloading a native code bearing application program 411 that provides system functionality through embedded native code. Native code is such programming instructions that the central processor 203 can execute directly without further processing. That is in contrast to the non-native instructions with which application programs 401 are typically written. Usually, application programs 401 are written in high-level languages such as Java or C# and compiled into an intermediate language such as Java Card bytecodes or IL-opcodes off-card prior to loading. These intermediate language codes are then loaded onto the card and interpreted by a virtual machine 409.

[37] The special native code bearing application programs 411 are downloaded onto the smart card 101 in the same manner as other application programs using the loader 405.

[38] The loader 405 and the interpreter 409 according to the invention is capable of detecting and processing native code bearing application programs 411. In a preferred embodiment the interpreter 409 interprets a language in which methods may be assigned attributes. An example of such a language is C# and discussions of method attributes may be found in any C# text book and many on-line tutorials on C#. In the preferred embodiment a method with embedded native code is given the NativeCode attribute with a parameter value being a text string equivalent to the opcodes for the native code implemented by the function.

The attribute `NativeCode` and its parameters form the body for the function that has the `NativeCode` attribute.

[39] The native code bearing application program 411 contains a particular function header for any method with embedded native code that is indicative to the loader 405 and to virtual machine 409 that the method contains embedded native code. In the preferred embodiment, native code bearing application 411 methods with embedded native code bear the header "internal extern static", which indicates to the virtual machine 409 that the method is a method with embedded native code. With respect to the loader 405, when it has detected that the native code bearing application 411 contains embedded code it finds in the load file the native code from the `NativeCode` attribute parameter. Similarly the virtual machine 409 detects from the method header that the method is a native code bearing method and that it should not attempt to interpret the body of the method but rather pass it onto the central processing unit 203 for execution in native mode. One way that the virtual machine 409 can pass control to the embedded code is to position the program counter of the central processor 203 at the memory location that the loader 405 placed the native code.

[40] Consider for example a function that will increment the value *i*.

Table 1 contains the assembly language for such a function:

Opcode	Instruction
<NativeIncrement>:	
39 12	ldw \$r2
66 01	mov \$r0,1
3C 12	stw \$r2

```

C0 21 29  add  $data0,$data1,$data1
78          ret

```

[41] Table 1. Assembly and Opcodes for Example Method

[42] (Note: this instruction set and opcodes correspond to the ST22L128

5 Smartcard 32-Bit RISC MCU from ST Microelectronics. However, the present invention is in no way limited to this particular processor and instruction set.)

[43] Thus, the opcodes for the ST22L128 for that code section is
 10 "391266013C12C0212978".

[44] Table 1 is an example of an application program 411 with embedded native code:

```

15      1  public int NativeCallTest (int val)
        2  {
        3      return NativeIncrement (val)
        4  }

        5  [NativeCode ("391266013C12C0212978")]
20      6  internal extern static int NativeIncrement (int
           i);

```

[45] Table 2. C# code for Application embedding the opcode for the Example Method.

25 [46] This example native code bearing application program 411 consists of two methods, namely, NativeCallTest and NativeIncrement. Whenever NativeCallTest is called, it calls NativeIncrement, which in turn is executed using the native code string provided. The virtual machine 409, when it encounters the call to NativeIncrement, redirects

the program counter to that portion of memory at which the string "391266013C12C0212978" has been stored by the loader 405.

[47] In a second aspect of the invention, the native code embedded into a native code bearing application program 411 is loaded into a specified address of the smart card system memory. Such an application is illustrated in Table 3.

```

1      [NativeCode (0xA6100, "XX ... XX")]
2      internal      extern      static      void
BiometricAlgorithmCorrection (void)
10 [48] Table 3. Embedded Native Code to be loaded at specific memory
location.
```

[49] As illustrated, in this alternative mechanism for embedding native code in an application, the NativeCode attribute has two parameters, namely, an address at which to load the embedded native code and a character string representation of the opcodes to load at that address.

[50] When the loader 405 encounters a native code bearing application 411 with a memory address parameter, the loader loads the corresponding native code string into the system memory beginning at that memory address. The native code bearing application 411 of Table 3 would be loaded at memory address 0xA6100. At run-time when the function BiometricAlgorithmCorrection() is called, the native code loaded at memory address is executed by the central processing unit 203.

[51] In a preferred embodiment, an additional security mechanism is implemented. Any entity that seeks to update the system software 403 is required to be authorized for perfecting such system updates by

downloading an application program with embedded native code 411. This check may be performed at two opportunities. First, by the loader 405 when the native code bearing application 411 is being loaded; second, by the virtual machine 409 at execution time. If the native code bearing application 411 has not been signed or has not been signed by an authorized entity, the application 411 is either rejected by the loader 405 or an error condition is flagged by the virtual machine 409 at run-time.

[52] In a preferred embodiment, the authorization that the originating entity has the requisite rights to perform the download of an application 411 with embedded native code is performed using Public Key Infrastructure (PKI) digital certificates. Alternatively, the security mechanism is performed using the GlobalPlatform application signature mechanism (Delegate Loading) or dynamic authentication schemes.

[53] The loader 405 and/or the virtual machine 409, upon detecting that an application program 411 has embedded therein native code, calls the PKI system 417 of the smart card 101 to verify that the application bears a digital signature from a trusted source. In a preferred embodiment, only the smart card manufacturer is deemed a trusted source. However, in alternative embodiments other trusted entities may be given the authority to access system level functionality by providing application program with embedded native code 411.

[54] Figure 5 is a flow chart illustrating the operations of the loader 405 according to the invention. The loader 405 operates to load the application program into the memory of the smart card and to resolve any unresolved

references. While performing the loading operations, if the loader 405 encounters an application with a NativeCode attribute, step 503, the loader knows that it has detected a method with embedded native code and proceeds to process the application as such, steps 505 through 513.

5 [55] A first step is to determine whether the entity that is attempting to download the application onto the smart card 101 may do so by determining whether the application has been signed by a trusted entity as described above, step 505. If the application 411 has not been signed by a trusted entity, the application 411 is rejected, step 506.

10 [56] If the application may be loaded onto the smart card 101, the loader proceeds with determining whether the native code is to be placed at a particular memory location, step 507. If the NativeCode attribute indicates that the native code of the native code bearing application being loaded is to be placed at a particular location, it places the code at the
15 specified memory location, step 509. If not, the loader 405 places the native code embedded in the application 411 into a new memory location, step 511.

[57] Having concluded placing the native code bearing application 411 into memory of the smart card 101, the loader can proceed with other
20 loader functions, step 513.

[58] Although specific embodiments of the invention has been described and illustrated, the invention is not to be limited to the specific forms or arrangements of parts so described and illustrated. For example, the

invention, while described in the context of smart cards for illustrative purposes, is applicable to other computing devices. The term *system software* is used throughout this document, however, smart card operating systems and other related modules may be embodied in ROM, non-volatile
5 memory, firmware, programmable logic arrays, etc. In reading the specification and interpreting the claims, all such implementations are to be included in the term *system software*. Similarly, both in the specification and the claims, the term *software* taken in isolation should also be interpreted in a broad sense to include firmware, etc. The
10 invention is limited only by the claims.

[59] CLAIMS

1. A method of updating system software of a resource-constrained
5 device having a microprocessor having associated therewith a
native code instruction set requiring no processing prior to
execution by the microprocessor, comprising:

providing functionality in the system software to determine
whether an application contains embedded native
10 code; and

in response to detecting that an application contains
embedded native code, passing the embedded native
code directly to the microprocessor for execution.
2. The method of Claim 1, wherein the step of passing the embedded
15 native code directly to the microprocessor comprises:

positioning the program counter at the embedded code.
3. The method of Claim 1, wherein the step of determining whether
an application contains embedded native code comprises detecting
the presence of a header indicative of the presence of native code in
20 the application.
4. The method of Claim 1, wherein the step of passing the embedded
native code directly to the microprocessor for execution comprises
moving the microprocessor program counter to a location where the
native code has been loaded into memory.
- 25 5. The method of Claim 1, further comprising:

associating an originating entity with the application; and

determining whether the originating entity is authorized to
update the system software by downloading an
application with embedded native code.

6. The method of Claim 5, wherein the originating entity is authorized if the originating entity is a trusted originating entity and the application containing embedded native code has been signed using a digital certificate.
- 5 7. The method of Claim 1 wherein the system software has a loader for loading application programs and an application containing embedded native code has a memory address parameter and wherein the loader in response to detecting that an application
10 program has a memory address parameter loads the embedded native code of the application containing embedded native code at an address specified by the address parameter.
8. A resource-constrained device having a processor having associated therewith a native code instruction set requiring no processing prior to execution by the microprocessor, comprising:
15 a memory connected to the central processing unit and containing application programs and system software for controlling operation of the microprocessor, the system software containing an operating system operable to execute the application programs loaded
20 into the memory, the operating system containing logic to detect that an application program includes embedded native code and logic to pass such embedded native code directly to the microprocessor for execution.
- 25 9. The resource-constrained device of Claim 8, wherein the logic to detect that an application program includes embedded native code comprises logic to detect that the application program contains a header indicative of that the application program contains embedded native code.

10. The resource-constrained device of Claim 8, wherein the logic to pass such embedded native code directly to the microprocessor for execution comprises logic to place the microprocessor program counter at the location where the native code has been loaded into memory.
11. The resource-constrained device of Claim 8, wherein an application program having embedded native code has an originating entity associated therewith, wherein the operating system further comprises logic to determine whether the originating entity is authorized to download application programs with embedded native code onto the resource-constrained device.
12. The resource-constrained device of Claim 11, wherein the originating entity is authorized if the originating entity is a trusted originating entity and the application containing embedded native code has been signed using a digital certificate.
13. The resource-constrained device of Claim 12 wherein the system software has a loader for loading application programs and an application containing embedded native code has a memory address parameter and wherein the loader in response to detecting that an application program has a memory address parameter loads the embedded native code of the application containing embedded native code at an address specified by the address parameter.

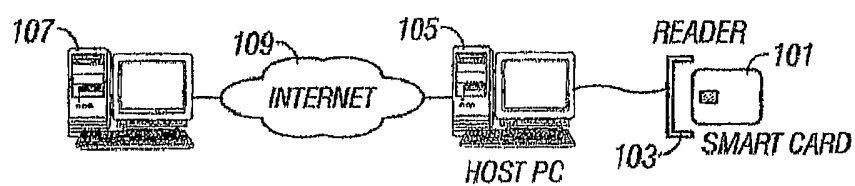


FIG. 1

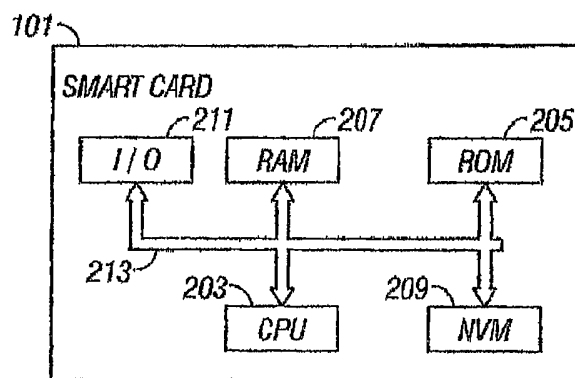


FIG. 2

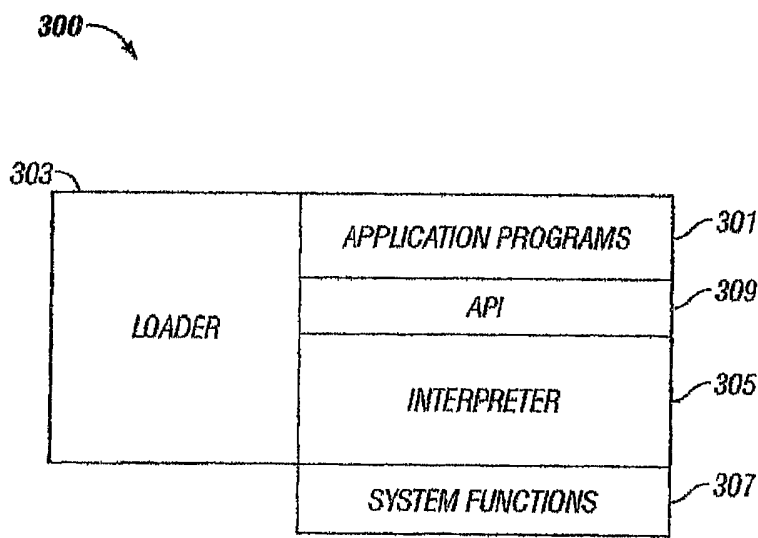


FIG. 3

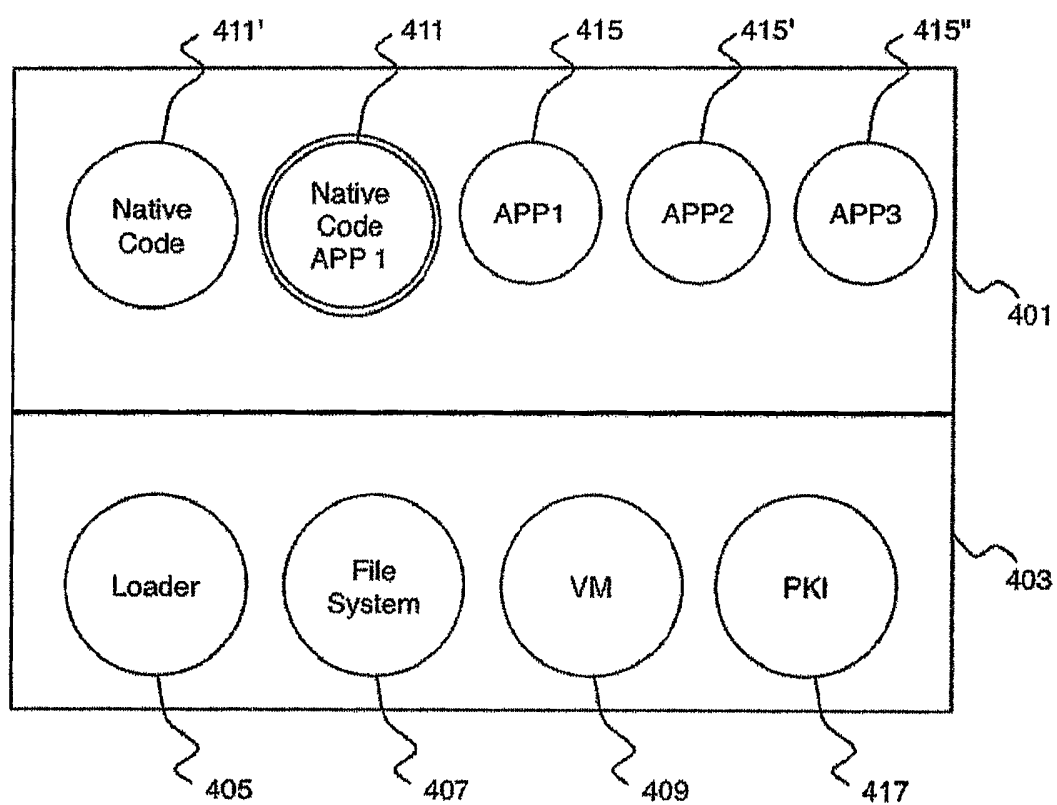
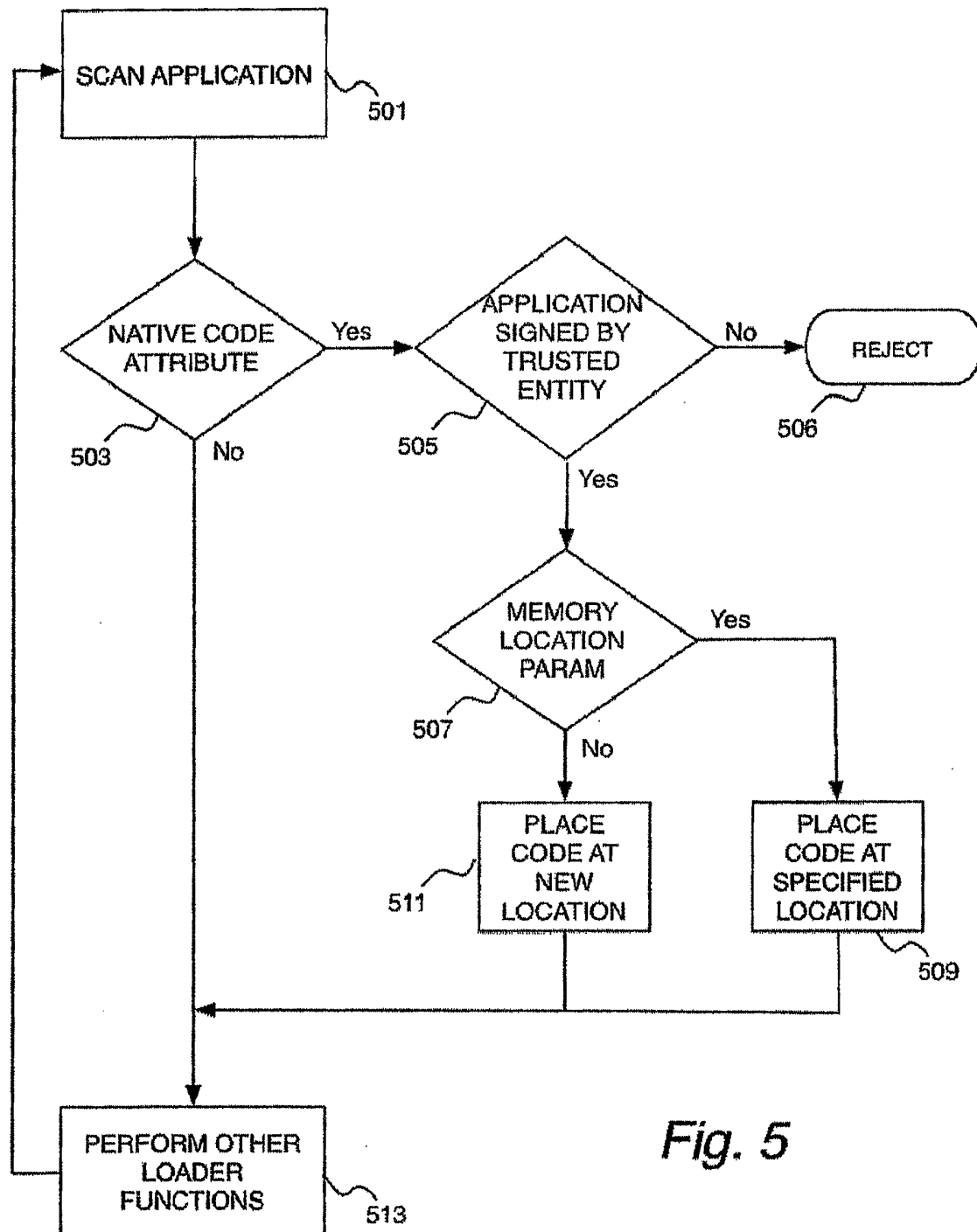


Fig. 4

*Fig. 5*

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB2005/002975

A. CLASSIFICATION OF SUBJECT MATTER

G07F7/10

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	TUAL J-P: "MASSC: A GENERIC ARCHITECTURE FOR MULTIAPPLICATION SMART CARDS" IEEE MICRO, IEEE SERVICE CENTER, LOS ALAMITOS, CA, US, vol. 19, no. 5, September 1999 (1999-09), pages 52-61, XP000862509 ISSN: 0272-1732 abstract page 55 figures 1,2 ----- -/--	1-13

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- * & * document member of the same patent family

Date of the actual completion of the international search

6 December 2005

Date of mailing of the international search report

20/12/2005

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Breugelmans, J

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB2005/002975

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6 308 317 B1 (WILKINSON TIMOTHY J ET AL) 23 October 2001 (2001-10-23) abstract column 1, lines 15-18,49-54 column 3, lines 3-16 column 3, line 50 - column 4, line 8 column 5, lines 1-9,30-40 column 8, lines 20-22 column 11, line 59 - column 12, line 64 column 13, line 28 - column 14, line 10 figure 16	1-13
X	----- RANKL WOLFGANG ET AL: "HANDBUCH DER CHIPKARTEN. AUFBAU - FUNKTIONSWEISE - EINSATZ VON SMART CARDS" HANDBUCH DER CHIPKARTEN. AUFBAU - FUNKTIONSWEISE - EINSATZ VON SMART CARDS, MUENCHEN : CARL HANSER VERLAG, DE, 1999, pages 191-193,252, XP002201839 ISBN: 3-446-21115-2 paragraph '5.10! paragraph '5.10.1!	1-13
A	----- WO 01/29791 A (TRESOR TV PRODUKTIONS GMBH; YUN, JAY; WILHELM, SIEGFRIED, E) 26 April 2001 (2001-04-26) abstract page 4, line 6 - page 5, line 3 page 9, line 4 - page 10, line 13 page 11, lines 10-26	1-13
A	----- US 6 390 374 B1 (CARPER TODD ET AL) 21 May 2002 (2002-05-21) abstract column 2, line 60 - column 3, line 7 column 4, lines 40-46 column 9, lines 40-48	1-13
A	----- EP 1 318 488 A (MATSUSHITA ELECTRIC INDUSTRIAL CO., LTD) 11 June 2003 (2003-06-11) abstract paragraph '0104!	1-13
A	----- US 6 328 217 B1 (EVERETT DAVID BARRINGTON ET AL) 11 December 2001 (2001-12-11) abstract column 5, lines 19-30 ----- -/--	1-13

INTERNATIONAL SEARCH REPORT

International Application No
PC 17 1B2005/002975

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	DAVID C. TOLL - IBM RESEARCH / PHILIPS SEMICONDUCTORS: "Secure Embedded Systems Project" INTERNET ARTICLE, 'Online! 20 December 2002 (2002-12-20), XP002357513 Retrieved from the Internet: URL: http://web.archive.org/web/20021220095007/http://www.research.ibm.com/secureos/ 'retrieved on 2005-12-05! the whole document	1-13
A	----- US 6 145 739 A (BERTINA ET AL) 14 November 2000 (2000-11-14) abstract column 3, lines 35-50 column 7, line 60 - column 8, line 20 column 9, lines 13-20, 39-46 column 11, lines 33-38 column 14, line 30 - column 16, line 65 -----	1-13

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/IB2005/002975

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 6308317	B1	23-10-2001	US 2003023954 A1	30-01-2003
WO 0129791	A	26-04-2001	NONE	
US 6390374	B1	21-05-2002	NONE	
EP 1318488	A	11-06-2003	CN 1423232 A	11-06-2003
			JP 2003173427 A	20-06-2003
			US 2003146277 A1	07-08-2003
US 6328217	B1	11-12-2001	AU 7776798 A	08-12-1998
			EP 0981807 A2	01-03-2000
			WO 9852158 A2	19-11-1998
			JP 2001525956 T	11-12-2001
US 6145739	A	14-11-2000	NONE	