US 20100257376A1

(19) **United States**

(12) **Patent Application Publication**

LITTLE et al.

(10) **Pub. No.: US 2010/0257376 A1**

(43) **Pub. Date: Oct. 7, 2010**

(54) **SYSTEM AND METHOD FOR MANAGEMENT OF PLAINTEXT DATA IN A MOBILE DATA PROCESSING DEVICE**

(76) Inventors: **Herbert A. LITTLE**, Waterloo (CA); **Anthony Scian**, Waterloo (CA)

Correspondence Address:
Dimock Stratton LLP/Research In Motion Limited
20 Queen Street West, 32nd Floor, Box 102
Toronto, ON M5H 3R3 (CA)

(21) Appl. No.: **12/818,649**

(22) Filed: **Jun. 18, 2010**

Related U.S. Application Data

(63) Continuation of application No. 11/221,196, filed on Sep. 6, 2005, now Pat. No. 7,783,896.

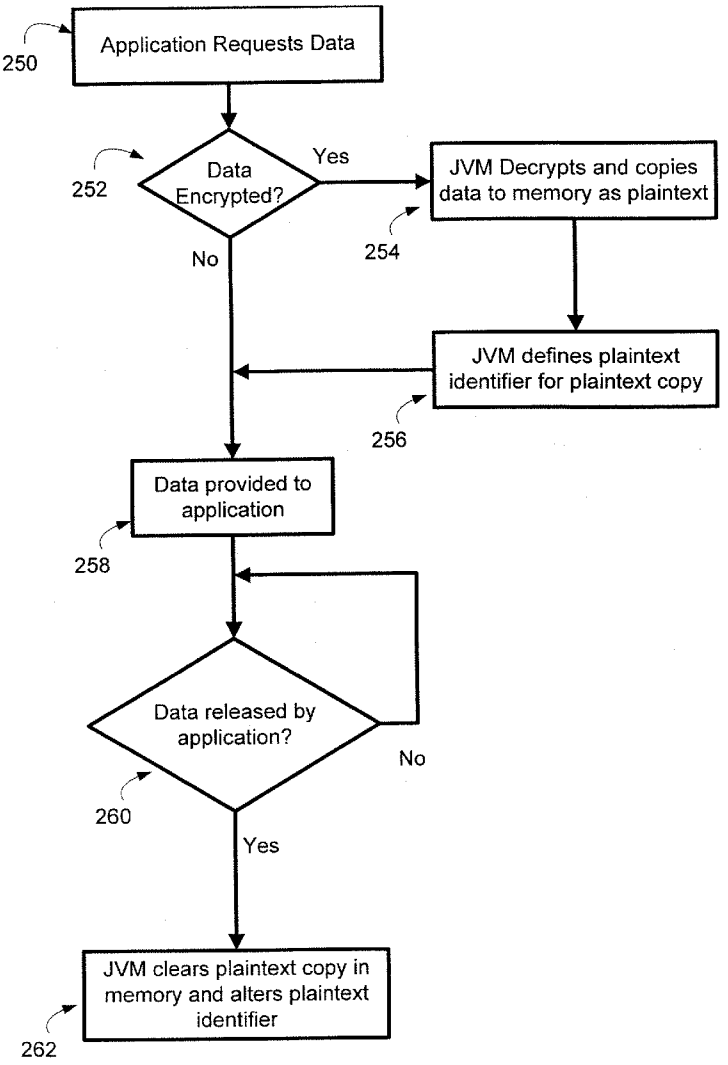
Publication Classification

(51) **Int. Cl.**
G06F 12/14 (2006.01)

(52) **U.S. Cl.** **713/189**

(57) **ABSTRACT**

A handheld data processing device includes stored data that is intended to be kept secure from unauthorized access. The handheld data processing device includes applications that store such secure data and which make use of plaintext data corresponding to the secure data. An identifier is defined to be associated with defined plaintext data. When the handheld data processing device is placed in a locked or secure state, code executable on the device is able to search for plaintext identifiers. Code executable on the device is consequently able to display to the user whether plaintext data is stored on the device or not.



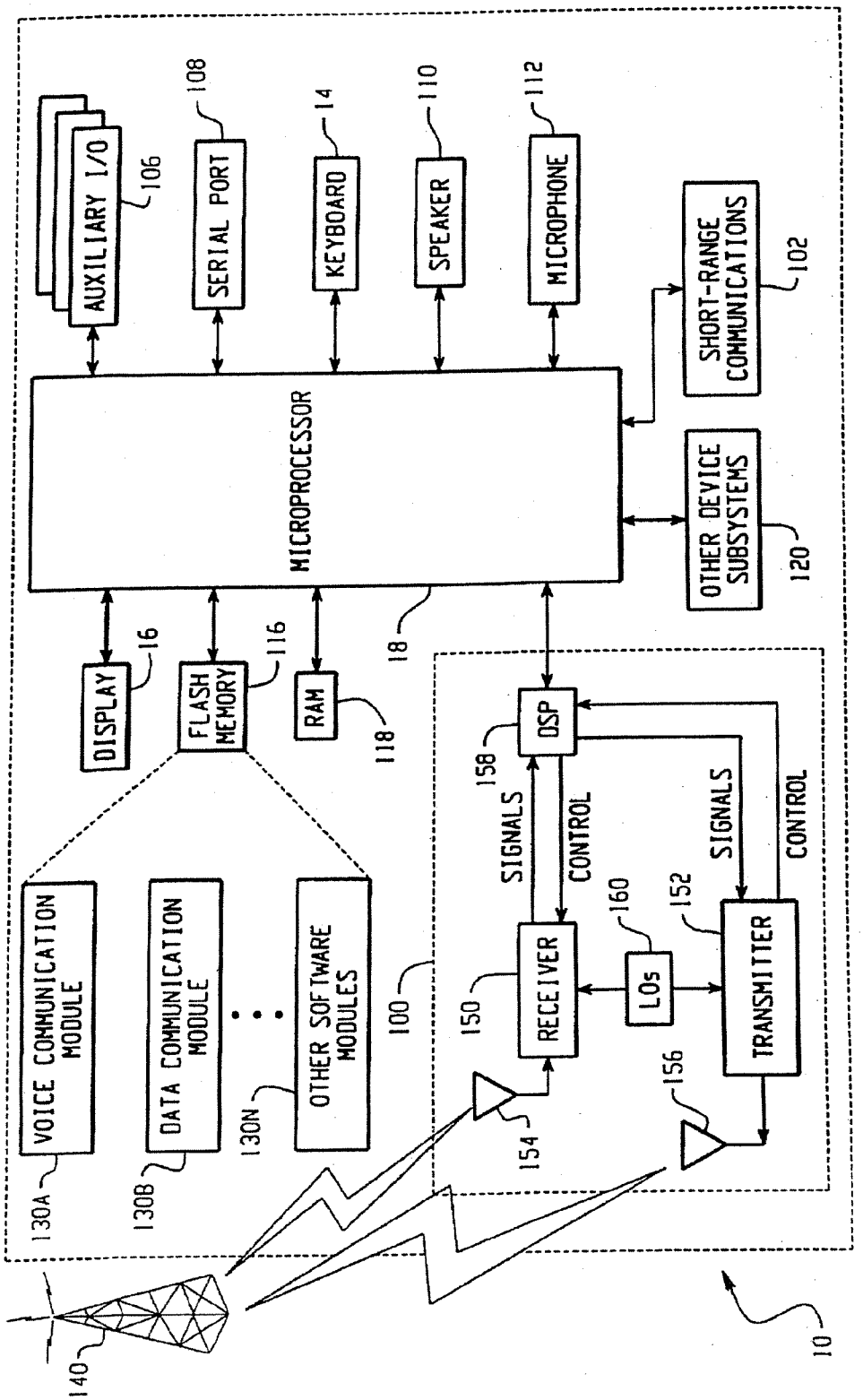

AUXILIARY I/O
106
SERIAL PORT
108
KEYBOARD
14
SPEAKER
110
MICROPHONE
112
SHORT-RANGE COMMUNICATIONS
102
MICROPROCESSOR
OTHER DEVICE SUBSYSTEMS
120
DISPLAY
16
FLASH MEMORY
116
RAM
118
18
DSP
158
SIGNALS
CONTROL
SIGNALS
CONTROL
VOICE COMMUNICATION MODULE
130A
DATA COMMUNICATION MODULE
130B
OTHER SOFTWARE MODULES
130N
100
150
RECEIVER
160
LOs
152
TRANSMITTER
154
156
140
10

FIG. 1

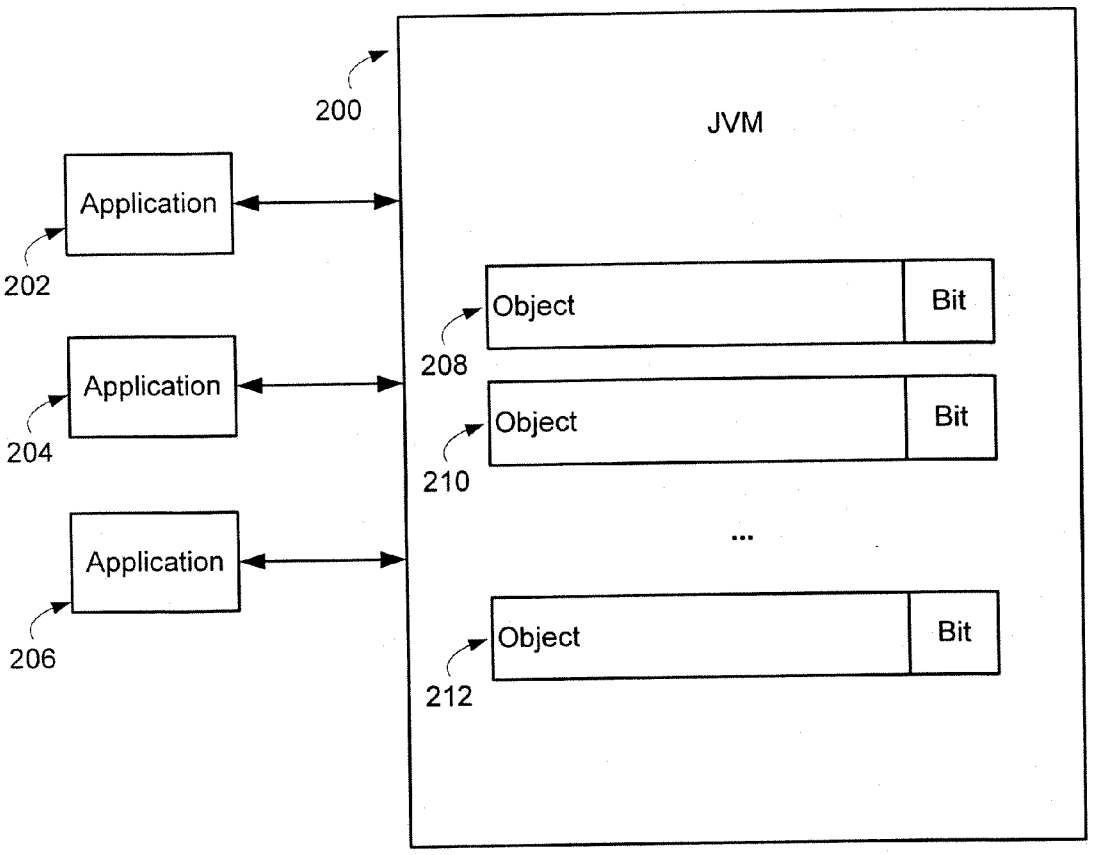

200
JVM
Application
202
Application
204
Application
206
Object
Bit
208
Object
Bit
210
...
Object
Bit
212

Figure 2

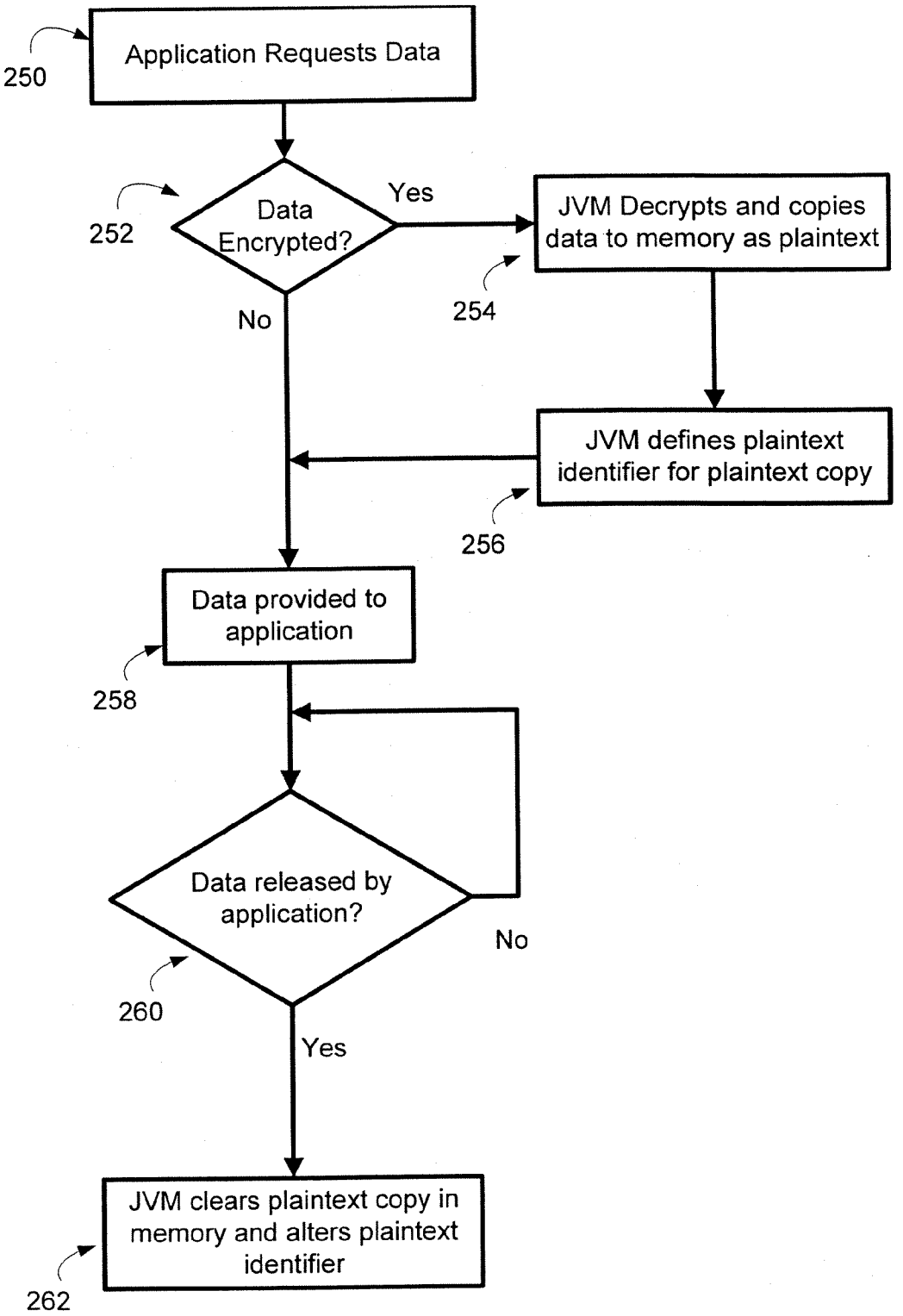

Application Requests Data
250
Data Encrypted?
252
Yes
No
JVM Decrypts and copies data to memory as plaintext
254
JVM defines plaintext identifier for plaintext copy
256
Data provided to application
258
Data released by application?
260
No
Yes
JVM clears plaintext copy in memory and alters plaintext identifier
262

Figure 3

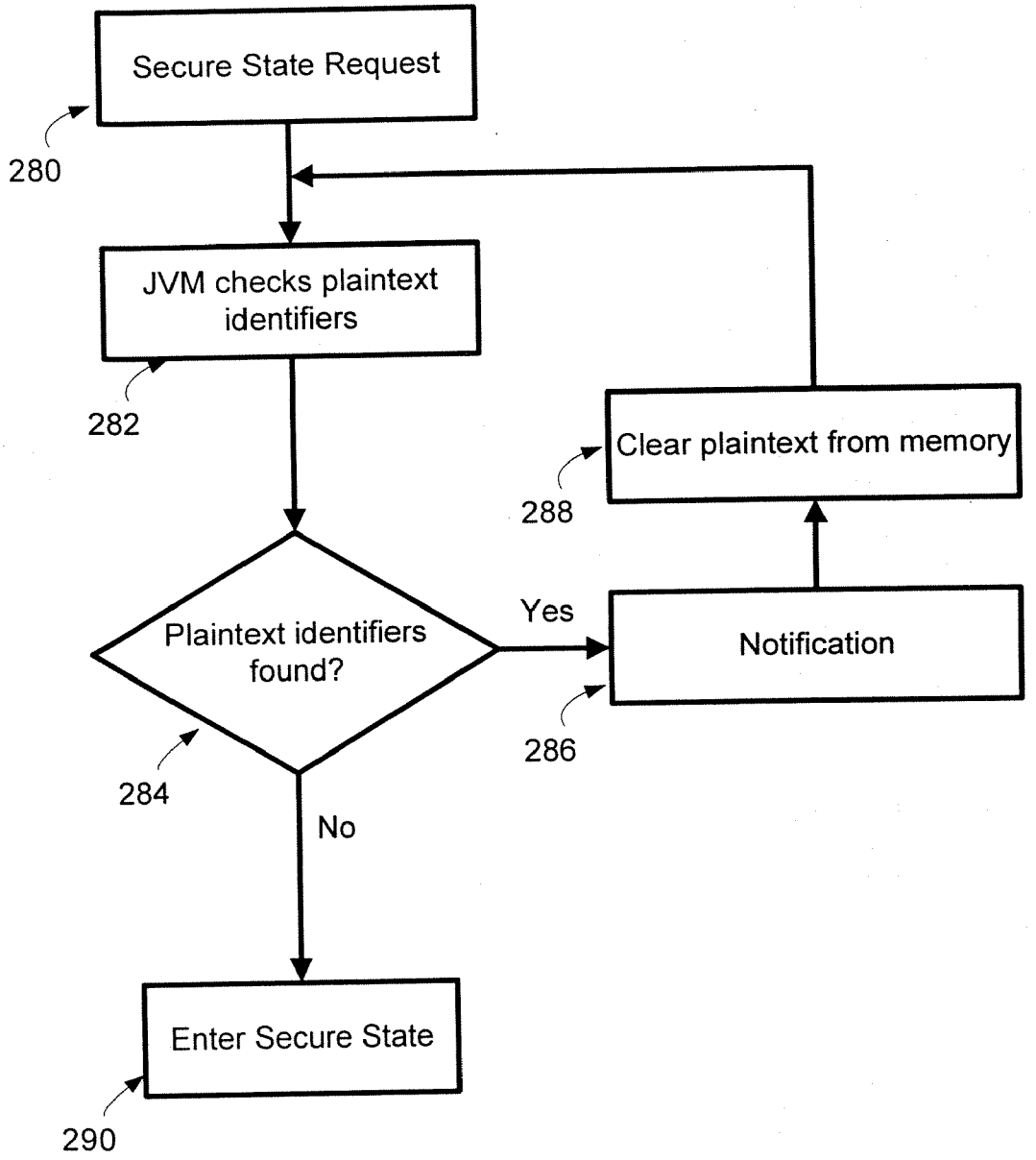

Secure State Request
280
JVM checks plaintext identifiers
282
Plaintext identifiers found?
284
Yes
No
Notification
286
Clear plaintext from memory
288
Enter Secure State
290

Figure 4

SYSTEM AND METHOD FOR MANAGEMENT OF PLAINTEXT DATA IN A MOBILE DATA PROCESSING DEVICE

REFERENCE TO PRIOR APPLICATIONS

[0001] This application is a continuation of U.S. application Ser. No. 11/221,196, filed Sep. 6, 2005.

FIELD OF INVENTION

[0002] This invention relates to mobile data processing systems and devices. In particular, this invention relates to a system and method for managing plaintext data in memory of a mobile data processing device.

BACKGROUND OF THE INVENTION

[0003] The use of mobile data processing devices has increased significantly in recent years. In addition to so called "laptop" and "tablet" computers, there is a growing popularity in handheld mobile data processing devices, sometimes called "personal digital assistants" or "PDAs" as well as smart phones. These mobile data processing devices are capable of storing a significant amount of user data, including calendar, address book, tasks and numerous other types of data for business and personal use. Most handheld data processing devices have the ability to connect to a personal computer for data exchange, and many are equipped for wireless communications using, for example, conventional email messaging systems. Depending upon the user's needs much of this data can be highly sensitive in nature, for example, where the device is used in government, the military or a commercial enterprise.

[0004] Because of their mobile nature, such devices may be lost or stolen with the consequential risk that data on the devices will be accessed by unauthorized individuals. For this reason, mobile data processing systems are typically password protected. However, such protection may be insufficiently secure if the data stored on the device is not encrypted. Accordingly, data stored in persistent memory on a handheld device is typically encrypted using an encryption key. For the data to be accessed by an application executing on the device, a decrypted copy of the encrypted data, or plaintext data, is made available for use by the application. The plaintext data may be used in processing carried out by the application, may be displayed on the device for viewing by the user, or may be sent to other users through email or other delivery means. Since the plaintext data is by definition not encrypted, an unauthorized user of the device may be able to view, copy or transmit an unencrypted copy of the data if such a user gains access to the device.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] In drawings which illustrate by way of example only a preferred embodiment of the system,

[0006] FIG. **1** is a block diagram of a system overview of a conventional mobile data processing device.

[0007] FIG. **2** is a block diagram showing an example relationship between the Java virtual machine, applications and plaintext data according to the preferred embodiment.

[0008] FIG. **3** is a flowchart showing steps in marking plaintext data and clearing plaintext data in accordance with an implementation of the preferred embodiment.

[0009] FIG. **4** is a flowchart showing steps for placing a device in a secure state in accordance with an implementation of the preferred embodiment.

DETAILED DESCRIPTION OF THE INVENTION

[0010] According to an aspect of the invention, when an application executing on a mobile data processing device decrypts secure data into plaintext, an identifier is created that is associated with that plaintext data and its location in memory. If an application creates copies of the plaintext data, or generates new plaintext data, additional identifiers are created to track the plaintext data in memory.

[0011] According to another aspect of the invention, when the mobile data processing device is "locked", or transferred to a secure state, a process executes which accesses memory in the device to identify any plaintext data in memory. A display is provided to the user to indicate whether or not there is plaintext data remaining in the device memory. Since there are identifiers associated with each piece of sensitive plaintext data, the process operating to access the memory of the device effectively evaluates the success of the device entering a secure state. This provides feedback to the user intended to prevent the user from considering the system to be secure when, in reality, there are still plaintext objects remaining in the device memory.

[0012] According to one aspect of the invention, there is provided a method of monitoring the use of decrypted plaintext data and ensuring that all plaintext data has been cleared from memory when the device enters a secure state.

[0013] An aspect of the present invention further provides a system for evaluating the performance of applications and ensuring that all plaintext data created by an application can be cleared from the system.

[0014] The present invention further provides a method of securing a device to ensure no decrypted data remains in memory after the device has entered the secure state.

[0015] A preferred embodiment of the system of the invention will be described in detail below, by way of example only, in the context of a hand-held mobile data processing device having wireless communications capabilities as illustrated in FIG. **1**. However, it will be appreciated that the principles apply to other data processing devices and the system is not intended to be limited thereby.

[0016] The hand-held data processing devices **10** include a housing, a keyboard **14** and an output device **16**. The output device shown is a display **16**, which is preferably a full graphic LCD. Other types of output devices may alternatively be utilized. A processor **18**, which is shown schematically in FIG. **1**, is contained within the housing and is coupled between the keyboard **14** and the display **16**. The processor **18** controls the operation of the display **16**, as well as the overall operation of the mobile device **10**, in response to actuation of keys on the keyboard **14** by the user.

[0017] The housing may be elongated vertically, or may take on other sizes and shapes (including clamshell housing structures). The keyboard may include a mode selection key, or other hardware or software for switching between text entry and telephony entry.

[0018] In addition to the processor **18**, other parts of the mobile device **10** are shown schematically in FIG. **1**. These include communications subsystem **100**; short-range communications subsystem **102**; keyboard **14** and display **16**, along with other input/output devices **106**, **108**, **110** and **112**; as well as memory devices **116**, **118** and various other device

subsystems **120**. Mobile device **10** is preferably a two-way RF communication device having voice and data communication capabilities. In addition, mobile device **10** preferably has the capability to communicate with other computer systems via the Internet.

[0019] Operating system software executed by the processor **18** is preferably stored in a persistent store, such as a flash memory **116**, but may be stored in other types of memory devices, such as a read only memory (ROM) or similar storage element. In addition, system software, specific device applications, or parts thereof, may be temporarily loaded into a volatile store, such as memory **118** which may be random access memory (RAM). Communication signals received by the mobile device may also be stored to memory **118**.

[0020] The processor **18**, in addition to its operating system functions, enables execution of software applications **130**A-**130**N on the device **10**. A predetermined set of applications that control basic device operations, such as data and voice communications **130**A and **130**B, may be installed on the device **10** during manufacture. In addition, a personal information manager (PIM) application may be installed during manufacture. The PIM is preferably capable of organizing and managing data items, such as e-mail, calendar events, voice mails, appointments, and task items. The PIM application is also preferably capable of sending and receiving data items via a wireless network **140**. Preferably, the PIM data items are seamlessly integated, synchronized and updated via the wireless network **140** with the device user's corresponding data items stored or associated with a host computer system.

[0021] Communication functions, including data and voice communications, are performed through the communication subsystem **100**, and possibly through the short-range communications subsystem. The communication subsystem **100** includes a receiver **150**, a transmitter **152**, and one or more antennas **154** and **156**. In addition, the communication subsystem **100** also includes a processing module, such as a digital signal processor (DSP) **158**, and local oscillators (LOs) **160**. The specific design and implementation of the communication subsystem **100** is dependent upon the communication network in which the mobile device **10** is intended to operate. For example, a mobile device **10** may include a communication subsystem **100** designed to operate with the Mobitex™, Data TAC™ or General Packet Radio Service (GPRS) mobile data communication networks and also designed to operate with any of a variety of voice communication networks, such as AMPS, TDMA, CDMA, PCS, GSM, etc. Other types of data and voice networks, both separate and integrated, may also be utilized with the mobile device **10**.

[0022] Network access requirements vary depending upon the type of communication system. For example, in the Mobitex and DataTAC networks, mobile devices are registered on the network using a unique personal identification number or PIN associated with each device. In GPRS networks, however, network access is associated with a subscriber or user of a device. A GPRS device therefore requires a subscriber identity module, commonly referred to as a SIM card, in order to operate on a GPRS network.

[0023] When required network registration or activation procedures have been completed, the mobile device **10** may send and receive communication signals over the communication network **140**. Signals received from the communication network **140** by the antenna **154** are routed to the receiver **150**, which provides for signal amplification, frequency down conversion, filtering, channel selection, etc., and may also provide analog to digital conversion. Analog-to-digital conversion of the received signal allows the DSP **158** to perform more complex communication functions, such as demodulation and decoding. In a similar manner, signals to be transmitted to the network **140** are processed (e.g. modulated and encoded) by the DSP **158** and are then provided to the transmitter **152** for digital to analog conversion, frequency up conversion, filtering, amplification and transmission to the communication network **140** (or networks) via the antenna **156**.

[0024] In addition to processing communication signals, the DSP **158** provides for control of the receiver **150** and the transmitter **152**. For example, gains applied to communication signals in the receiver **150** and transmitter **152** may be adaptively controlled through automatic gain control algorithms implemented in the DSP **158**.

[0025] In a data communication mode, a received signal, such as a text message or web page download, is processed by the communication subsystem **100** and is input to the processor **18**. The received signal is then further processed by the processor **18** for an output to the display **16**, or alternatively to some other auxiliary I/O device **106**. A device user may also compose data items, such as e-mail messages, using the keyboard **14** and/or some other auxiliary I/O device **106**, such as a touchpad, a rocker switch, a thumb-wheel, or some other type of input device. The composed data items may then be transmitted over the communication network **140** via the communication subsystem **100**.

[0026] In a voice communication mode, overall operation of the device is substantially similar to the data communication mode, except that received signals are output to a speaker **110**, and signals for transmission are generated by a microphone **112**. Alternative voice or audio I/O subsystems, such as a voice message recording subsystem, may also be implemented on the device **10**. In addition, the display **16** may also be utilized in voice communication mode, for example to display the identity of a calling party, the duration of a voice call, or other voice call related information.

[0027] The short-range communications subsystem enables communication between the mobile device **10** and other proximate systems or devices, which need not necessarily be similar devices. For example, the short-range communications subsystem may include an infrared device and associated circuits and components, or a Bluetooth™ communication module to provide for communication with similarly-enabled systems and devices.

[0028] As will be appreciated by those skilled in the art, a device such as that represented by device **10** in FIG. **1** may be a wireless handheld device on which confidential information is stored and processed. In such a typical case, confidential information will be maintained as data for which access will be provided to authorized users, only. In the hand-held mobile data processing device of the preferred embodiment, different types of confidential information are stored on the device as encrypted data in memory **116**.

[0029] As is set out schematically in FIG. **2**, in the preferred embodiment, one of the software modules **130**A . . . **130**N is Java Virtual Machine **200** (JVM **200**). JVM **200** is a virtual machine that operates to interpret Java code comprising applications and to carry out related functions such as memory management, all in a manner understood for virtual machine function, and known to those skilled in the art. In the

preferred embodiment, all applications executable on the mobile device comprise code that is interpreted by JVM **200** and therefore device **10** may rely on the operation of JVM **200** to manage device memory **118**. As will be appreciated, other mobile processing devices may be implemented in which memory management is carried out in other ways. Although the description of the preferred embodiment relates to the use of JVM **200**, it will be appreciated that the memory management techniques set out will apply to other, non-Java implementations or to implementations where a virtual machine and other computing device code together carry out memory management.

[0030] FIG. **2** shows a block diagram illustrating the relationship between an example JVM **200** and applications that are interpreted by JVM **200** (shown as applications **202**, **204**, **206** in the example in the figure). Applications define objects which are managed by JVM **200**, shown by example in FIG. **2** as objects **208**, **210**, **212**. In typical operation, applications **202**, **204**, **206** will define and use objects **208**, **210**, **212**, as appropriate. JVM **200** will carry out memory management tasks relating to those objects to ensure that JVM **200** makes efficient use of the memory resources available on device **10**. This is particularly important for this mobile device environment where there may be significant constraints on the memory resources available in the device.

[0031] In the preferred embodiment, JVM **200** supports encryption of Java objects. If one of applications **202**, **204**, **206** requests an object that includes data stored in encrypted memory, JVM **200** is responsible for copying the encrypted data into its plaintext form and for then making the plaintext data available to the application (passing the data to the application). In typical operation, after decryption of the data in the requested object, the plaintext copy of the data is stored in memory **118** on device **10** so as to be available to the requesting application. In the preferred embodiment, applications **202**, **204**, **206**, for example, are operative to request data objects from JVM **200** and to release data objects using JVM **200**.

[0032] In a mobile data processing device, such as that on which the preferred embodiment is implemented, it is desirable for the memory management functionality of the device to be carried out to assure a given level of security for information stored as data on the device. For example, it is desirable to be able to place the device in a secure state. In a mobile data processing device, the user will typically use applications executing on the device to process data that is confidential. During the time that the user is making use of such data (using an email application to read and send email messages, making appointments in a calendar application, and so on) the device uses a plaintext copy of otherwise encrypted data.

[0033] However, when the user has completed making use of the device, it is advantageous to put the device in a secure state in which data on the device is in an encrypted form, only (and hence unavailable to unauthorized users). When a device enters such a secure state, the plaintext copies of encrypted data representing private information for the user that were used by applications on the device, such as a task list or calendar, are cleared from memory. In the preferred embodiment, a memory cleaner application is operative to send messages to running applications on the device to invoke steps in the application operation to remove copies of plaintext objects used by the applications. In such an arrangement each application is relied upon to correctly act on receiving such a message. If an application does not carry out these steps to clean up plaintext objects correctly, the device may enter what is intended to be a secure state with plaintext data in memory.

[0034] Further, if such steps are not correctly taken, the plaintext copies of data may be available to persons other than the authorized user of the device. Being able to switch between states where the plaintext data is available (for efficient device operation), and a state where it is not, is particularly important with a mobile data processing device where the use is intermittent and unpredictable, and where the potential for loss or theft of the device is relatively high.

[0035] In the preferred embodiment, as a part of the memory handling carried out by JVM **200**, an identifier is created and associated with the plaintext data to indicate that the data is either derived from encrypted data or the data is intended to be dealt with in a secure fashion on the device. Plaintext data is typically obtained after a decryption step. However, data which is intended to be encrypted at a future stage in application operation, or data which is temporarily used by an application but is intended to be kept secure, may also be considered to be plaintext data.

[0036] Different implementations of this aspect of the memory handling in JVM **200** are possible. In FIG. **2** one implementation is shown in which a predetermined bit within a byte array is defined to act as a flag. This is shown by the bits defined in objects **208**, **210**, **212**, respectively. When the appropriate bit is set, it identifies the associated byte array as representing plaintext data.

[0037] In another implementation of the preferred embodiment, the identifier is a short byte, or word, within a byte array. The byte or word may take different values to indicate not only that the associated data object is plaintext data but also to indicate which one of a set of possible levels of security is associated with the data. For instance, a word identifier may specify one of a set of public, private and secure data types for the associated object. In this implementation of the preferred embodiment, JVM **200** is able to invoke different memory management steps, depending on the data type specified by the value of the word identifier.

[0038] The JVM **200** of the preferred embodiment includes computing device code operative, by execution on the mobile device, to define a plaintext identifier, as described above, and to carry out memory management operations based on the value of a particular plaintext identifier. It should be noted that in this description of the preferred embodiment, there are several memory management steps and procedures that are described relating to the marking of plaintext by the use of a plaintext identifier. However, it will be appreciated that there may also be implementations where the marking of plaintext itself may be of use to applications themselves in their use of such data on the mobile data processing device. In such a case, the memory management component of JVM **200** relating to the plaintext identifiers created may be relatively restricted or minimal in its operation.

[0039] In the preferred embodiment, it is desirable to permit different types of plaintext memory objects to be retained as plaintext for different time periods. For example, the private information of the user such as the plaintext data representing the information in a task list, may be kept by the device in a plaintext form until the user specifies that the device should be placed in a secure (or locked) state. In another example, the device **10** of the preferred embodiment supports pass codes. The system and method of the preferred embodiment permits memory management of JVM **200** to clear the plaintext copy of an encrypted pass code from

memory as soon as the application that requested the decryption of the pass code has released it. This may happen before there is a request that the system enter a secure state.

[0040] As referred to above, although applications **202**, **204**, **206** are intended to function such that plaintext objects are not made available after release by the applications, through poor design or errors in the applications or by deliberate choice, such plaintext copies may be left in memory by an application. The approach of the preferred embodiment permits JVM **200** to manage the memory objects **208**, **210**, **212** such that on entering a secure state the user of the device will be notified as to whether there is plaintext data in memory.

[0041] In the preferred embodiment a ribbon application is defined to manage a portion of the content shown on display **16**. On display **16** a defined region contains a ribbon of information that relates to system usage and availability. The ribbon application is code executable on device **10** operable to obtain device status information and to manage the display of such information on display **16**. In the preferred embodiment, when there is a request to place device **10** in a secure state, the ribbon application makes use of JVM **200** to determine if there are plaintext objects in memory. JVM **200** has code executable to receive a request for the plaintext object status from the ribbon application. JVM **200** carries out the steps described below to determine if a plaintext object is in memory. If such a plaintext object is found, the appropriate message is returned to the ribbon application. The ribbon application operates to display an icon in the ribbon region of display **16** reflecting whether device **10** is in a secure state in which there are no plaintext objects available in memory (a closed padlock icon) or not (an open padlock icon).

[0042] In this manner, the use of plaintext bits permits the device user to be assured that the device in a secure state has no data that may be placed at risk or, alternatively, that application error has resulted in the device containing plaintext data in memory despite the attempt to make the device secure.

[0043] FIG. **3** shows a simple flowchart setting out the steps of the method carried out by the code of JVM **200** operative to ensure that plaintext copies of encrypted data are cleared when an application releases objects including encrypted data. At block **250**, an application request for access to a data object is made. In the preferred embodiment, applications are described as requesting access to data objects. Such access includes the step of the application defining a data object for subsequent use.

[0044] At decision block **252**, JVM **200** determines whether the data requested is encrypted. If the data is determined to be encrypted, at block **254** JVM **200** carries out the decryption (either directly or by launching a decryption process). Block **256** represents the step of JVM **200** defining a plaintext identifier for association with the plaintext copy of the encrypted data. The plaintext identifier may be a bit or set of bits (byte or word) associated with the byte array defining the plaintext data or, as described below, it may be a table entry associated with the plaintext copy of the data. JVM **200** saves a plaintext copy of the encrypted data to device memory (block **254**) for use by the requesting application. As is indicated in block **258** of FIG. **3**, JVM **200** provides the plaintext to the requesting application (in the preferred embodiment by specifying the availability of plaintext data in the device memory).

[0045] Decision block **260** shows JVM **200** executing to carry out the step of determining whether the requesting application has released the object having plaintext data associated with it. Although FIG. **3** shows a looping decision step, implementations of the system and method of the preferred embodiment will likely implement other methods for applications to specify the release of data objects to JVM **200**. Such methods are known to those skilled in the art and are not described further here. The result of the release of the object by the application is that, as shown in block **262**, the code of JVM **200** is operative to clear the plaintext data from device memory **118** and is operative to alter the plaintext identifier to reflect the removal of the plaintext data from device memory **118**.

[0046] As will be appreciated, where an implementation of the preferred embodiment defines the plaintext identifier as a specified bit or set of bits associated with a byte array in memory, the step of clearing the plaintext data from memory may include the step of similarly clearing the plaintext bit or bits from memory. Alternatively, if the plaintext identifier is defined in a table, as described below, JVM **200** may, as part of the step shown in block **262** of FIG. **3**, modify the value of the plaintext identifier (to show that the plaintext data previously in the device memory has been cleared) or may remove the table entry entirely.

[0047] As indicated above, a plaintext identifier mechanism that may be implemented as an alternative to a bit associated with a bitsteam in memory is a table that records all secure objects that have been decrypted. In the system of the preferred embodiment, such a table will be defined and maintained by JVM **200**. As will be appreciated by those skilled in the art, although this description describes a table with table entries, other similar data structures may be used to allow the relevant plaintext information to be stored and accessed, as described.

[0048] With respect to the table of plaintext identifiers, each entry in the table records the existence in the device memory of plaintext data that is a copy of encrypted data in the device, as well as the location in memory of the plaintext data. Optionally, the table entry includes information about the level of security for that object. An advantage of using a table to record plaintext identifiers is that for such an arrangement, JVM **200** does not carry out a scan through the device **10** memory **118** to locate plaintext identifiers such as the bits in objects **208**, **210**, **212** when determining whether to alter a plaintext identifier or determining whether there are plaintext data objects in device memory **118**. Instead, JVM **200** manages the table and as applications release objects, JVM **200** clears the plaintext data from memory and updates the table accordingly.

[0049] For a high-security implementation, FIG. **4** presents a simple flowchart for JVM **200** code operative to carry out the steps of placing device **10** in a secure state. In the example of FIG. **4**, device **10** is seeking to enter the secure state (block **280** shows receipt by JVM of a secure state request). As shown in block **282**, JVM **200** reviews the identifier table (rather than device memory **118** itself) to determine whether there are any plaintext identifier entries that correspond to plaintext data in device memory **118**. If there are such identifiers present in the table (decision block **284**) then JVM **200** will, in one embodiment of the preferred embodiment, notify the user (block **286**). This reporting mechanism may also indicate whether objects were improperly used by an application so as to create a potential security breach. This permits users to evaluate third party applications to ensure that they are maintaining information in a secure state. Without this

reporting mechanism, users may not know how applications are treating their confidential data.

[0050] Following, or in conjunction with, the notification step of block **286**, JVM **200** arranges to clear from memory the plaintext data referenced by the located plaintext identifier in the table (block **288**). This process of checking for plaintext identifiers, notifying the user and clearing the plaintext data is repeated until there are no further plaintext identifiers in the table. As will be appreciated, in one implementation it is possible for plaintext identifier entries to remain in the table even after the related plaintext data has been cleared. In such a case the entry will include a field to specify whether the entry is current or non-current. Non-current entries are potentially useful in assessing application or system performance or for troubleshooting application behaviour.

[0051] After all plaintext entries in the plaintext identifier table have been processed, JVM **200** is able to display information to assure the user that all confidential plaintext data has been released by the applications and cleared from memory. As described above, JVM **200** permits the device to display an icon or other message to report to the user that all plaintext copies of secure confidential information have been cleared from memory. The user can be assured that only encrypted confidential data is left on the device.

[0052] In the case where the JVM **200** determines that plaintext data has not been properly cleared from memory by application operation, various corrective measures are employed depending upon the security level of the data and of the device. Since applications use JVM **200** to store copies of secure objects in device memory **118**, JVM **200** acts as a gate keeper. In the preferred embodiment, JVM **200** will, if plaintext data mishandling is identified for an application, refuse to transfer a requested plaintext object to the application's database (potentially copied into device memory **118**). JVM **200** may report the presence of objects that have not yet been released by the application as is described above. If the application fails to clear the plaintext data for an object, JVM **200** is configurable to either clear the object itself or to report the presence of the plaintext object to the user, or both.

[0053] Depending upon the security level chosen for the device, a user can require JVM **200** to overwrite any remaining sensitive plaintext objects with null characters, or force a "reset" of the device to clear all sensitive plaintext information from memory before the device can enter the secure state. The advantage of this option is for high-security applications where an administrator would rather risk the device being unstable, through clearing memory without releasing objects, than permit sensitive data to remain on the device in plaintext form.

[0054] The use of a plaintext identifier table is also advantageous as JVM **200** may permit the definition of objects with defined security levels, as referred to above. Thus an entry in the plaintext identifier table will be able to specify the security level for the decrypted object data that is stored in plaintext form in the device memory. In this way, different reporting procedures and memory management steps may be carried out by JVM **200**, depending on the defined security level for device **10** and on the specified security level in the table entry for the data object. A simple example relating to different timing of memory management steps for pass codes has been given above.

[0055] Further, plaintext bits or a plaintext identifier may be used to prevent plaintext data from being written to flash memory **116** in device **10**. Because data which is tagged as plaintext by the plaintext identifier is intended to remain secure (either through encryption or deletion) JVM **200** may be defined to operate to preclude any application from writing plaintext data to persistent memory where the data may become available to unauthorized users. In one implementation, JVM **200** will lock if an application seeks to write plaintext data to persistent memory such as flash memory **116**. Where each data object has a plaintext bit associated with it, JVM **200** operates to inspect the plaintext bit when an application seeks to store the plaintext object in flash memory **116**. The status of the plaintext bit will determine whether JVM **200** writes the object to flash memory **116** or alternatively throws an exception or locks the device. In the preferred embodiment, the response for JVM **200** may be determined by policies set by the device administrator.

[0056] A further use of the plaintext identifier is in a debugging mode on device **10**, used by application developers. The preferred embodiment provides that the debugger application operates to provide information to application developers as to the number and potentially identity of plaintext objects in memory at different stages in application execution.

[0057] Various embodiments of the system and method of the invention having been thus described by way of example, it will be apparent to those skilled in the art that variations and modifications may be made without departing from the invention. For example, code adapted to provide the systems and methods described above may be provided on many different types of computer-readable media including computer storage mechanisms (e.g., CD-ROM, diskette, RAM, flash memory, computer's hard drive, etc.) that contain instructions for use in execution by a processor to perform the methods' operations and implement the systems described herein.

We claim:

1. A method that provides for a secure state on a mobile device, the method comprising:

generating plain-text data from encrypted data when the encrypted data is requested by an application on the mobile device;

retrievably storing the decrypted plain-text data in memory on the mobile device;

generating an identifier, the identifier associable with the generated plain-text data and comprising an indicator that the associated plain-text data is confidential;

detecting an indication that the mobile device is to enter a secure state;

retrieving and using the identifier to identify the associated plain-text data as confidential and delete the associated plain-text data from the mobile device memory; and

using a next identifier to identify and delete a next confidential plain-text data, if a next identifier comprising confidential plain-text data exists, until no undeleted confidential plain-text data is identified using identifiers.

2. The method of claim **1** further comprising, after retrieving the identifier, displaying an indication that confidential plain-text data exists in the memory of the mobile device, and, when there is no undeleted confidential plain-text data identifiable using identifiers, one of: undisplaying the indication, displaying a next indicator there is no more confidential plain-text data in memory, or, both undisplaying the indication and displaying the next indicator.

3. The method of claim **1** wherein storing the plaintext data in association with the identifier comprises storing the plain-text data and the associated identifier in a data object in the device memory.

4. The method of claim **1**, wherein the identifier comprises at least one bit in a byte array associated with the plaintext data.

5. The method of claim **1**, further comprising clearing plaintext data associated with a predetermined security level in response to a request from an application to release the plaintext content prior to receiving the request for the mobile data processing device to enter the secure state.

6. The method of claim **1**, further comprising, upon receipt of a request from an application to write plaintext data stored in the device memory to a persistent memory, writing the requested plaintext data to the persistent memory upon determining that the requested plaintext data is not associated with an identifier indicating that the plaintext data is intended to be treated in a secure fashion.

7. The method of claim **1** wherein identifies are table entries.

8. A method that provides for a secure state on a mobile device, the method comprising:

generating plain-text data from encrypted data if the encrypted data is requested by an application on the mobile device;

retrievably storing the decrypted plain-text data in memory on the mobile device;

generating an identifier, the identifier associable with the generated plain-text data and comprising an indicator that the associated plain-text data is security related;

generating a next identifier associable with plain-text data generated by the application, if and when there is any plain-text data generated by the application to be classified as security related, the next identifier comprising an indicator that the associated plain-text data is security related;

detecting an indication that the mobile device is to enter a secure state;

using the identifier and any next identifiers to determine if there is any associated plain-text data that is security related; and

resetting the mobile device if it was determined there is security related plain-text data.

9. The method of claim **7** where indicators further comprise a security level.

10. The method of claim **8** where the resetting of the mobile device only occurs when at least one identifier comprises a preselected security level.

11. A method that provides for a secure state on a mobile device, the method comprising:

generating plain-text data by an application running on the mobile device, or, generating plaintext data from encrypted data when the encrypted data is requested by the application;

retrievably storing the plain-text data in memory on the mobile device;

generating an identifier, the identifier associable with the generated plain-text data and comprising an indicator that the associated plain-text data is security related;

detecting an indication that the mobile device is to enter a secure state;

retrieving and using the identifier to identify the associated plain-text data that is security related and delete the security related plain-text data from the mobile device memory; and

using a next identifier to identify and delete a next confidential plain-text data, if a next identifier comprising security related plain-text data exists, until no undeleted confidential plain-text data is identified using identifiers.

12. The method of claim **11** further comprising, after retrieving the identifier, displaying an indication that confidential plain-text data exists in the memory of the mobile device, and, when there is no undeleted confidential plain-text data identifiable using identifiers, one of: undisplaying the indication, displaying a next indicator there is no more confidential plain-text data in memory, or, both undisplaying the indication and displaying the next indicator.

13. The method of claim **11** wherein storing the plaintext data in association with the identifier comprises storing the plaintext data and the associated identifier in a data object in the device memory.

14. The method of claim **11** wherein the identifier comprises at least one bit in a byte array associated with the plaintext data, or, the identifiers are table entries.

15. The method of claim **11** further comprising, upon receipt of a request from an application to write plaintext data stored in the device memory to a persistent memory, writing the requested plaintext data to the persistent memory upon determining that the requested plaintext data is not associated with an identifier indicating that the plaintext data is intended to be treated in a secure fashion.

16. The method of claim **11** where the security level of an indicator further comprising a plurality of security levels, and the clearing plaintext data associated with the indicator depends on the security level of the indicator.

* * * * *