US 20220116213A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2022/0116213 A1**

Pala et al. (43) **Pub. Date: Apr. 14, 2022**

(54) **METHOD AND APPARATUS FOR MANAGING CRYPTOGRAPHIC KEYS**

(71) Applicant: **Robert Bosch GmbH**, Stuttgart (DE)

(72) Inventors: **Diego Pala**, Bochum (DE); **Teona Tatovic**, Bochum (DE); **Lukas Riemenschneider**, Shanghai (CN)

(21) Appl. No.: **17/448,277**

(22) Filed: **Sep. 21, 2021**

(30) **Foreign Application Priority Data**

Oct. 9, 2020 (DE) 102020212772.7

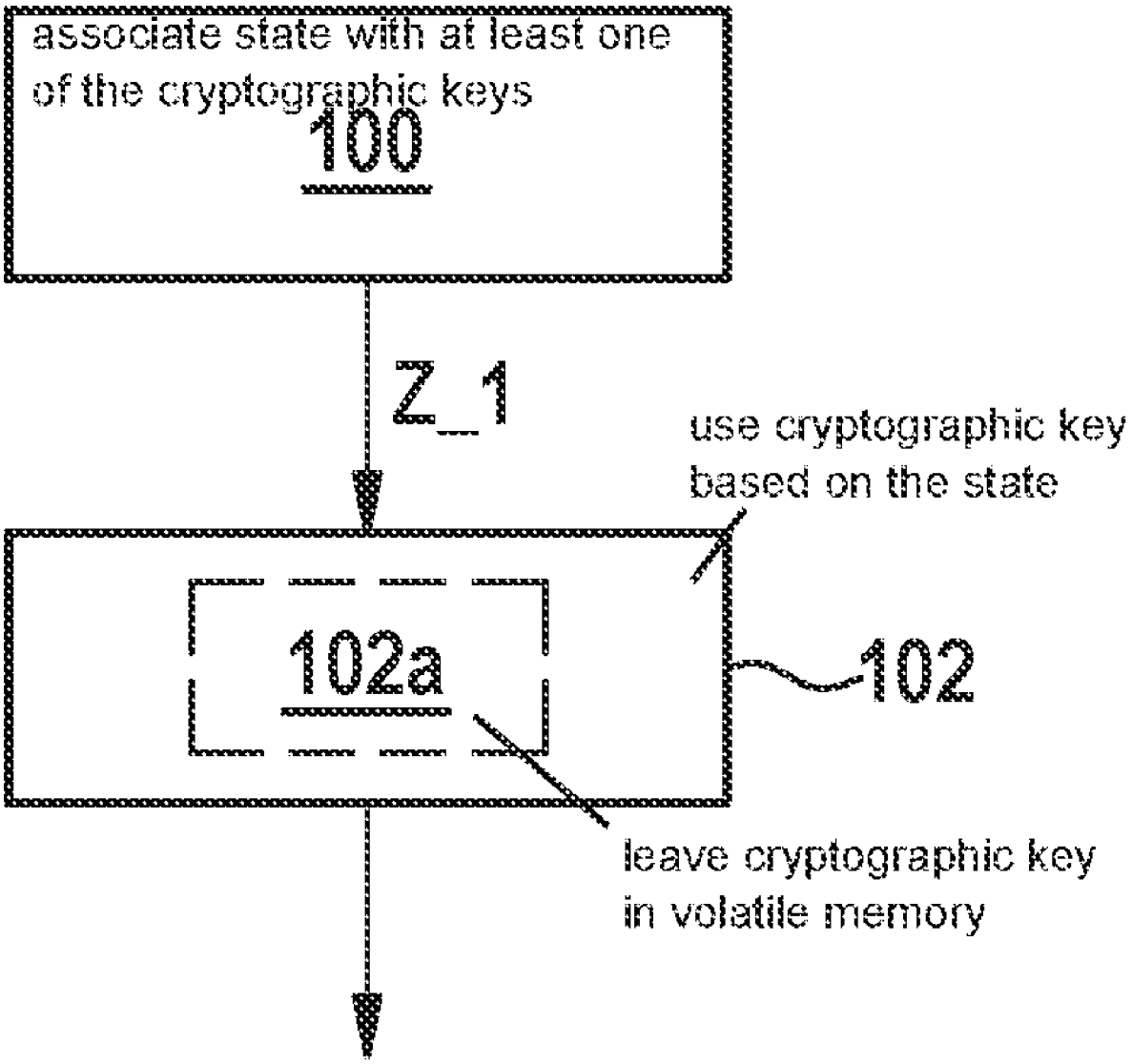
Publication Classification

(51) **Int. Cl.**
H04L 9/08 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 9/0894*** (2013.01); *H04L 2209/84* (2013.01)

(57) **ABSTRACT**

A method for managing cryptographic keys for a control device, in particular for a motor vehicle. The method includes the following steps: associating a state with at least one of the cryptographic keys; using the at least one cryptographic key based on the state.



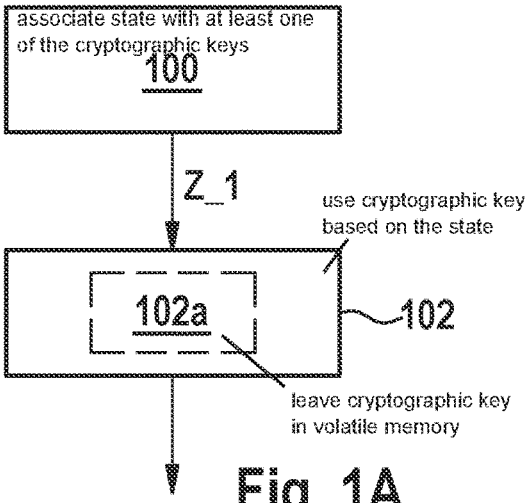


Fig. 1A

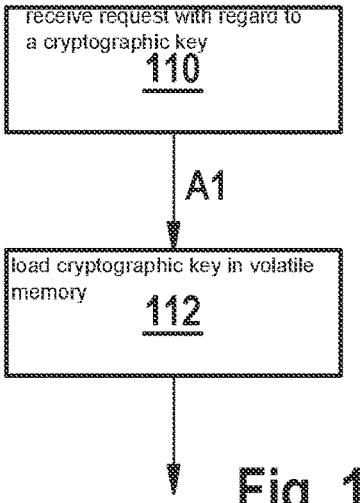


Fig. 1B

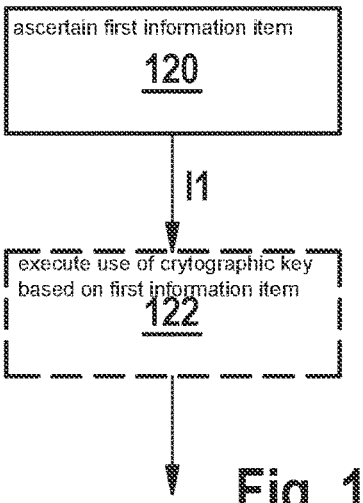


Fig. 1C

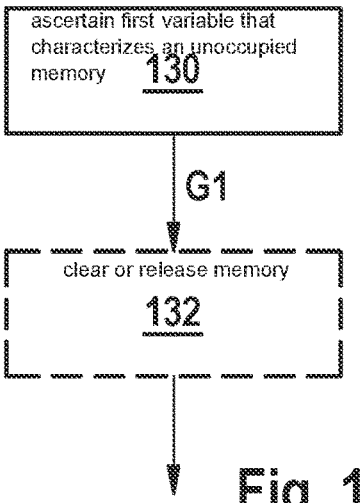


Fig. 1D

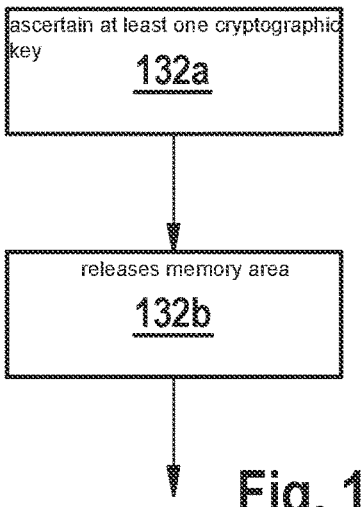


Fig. 1E

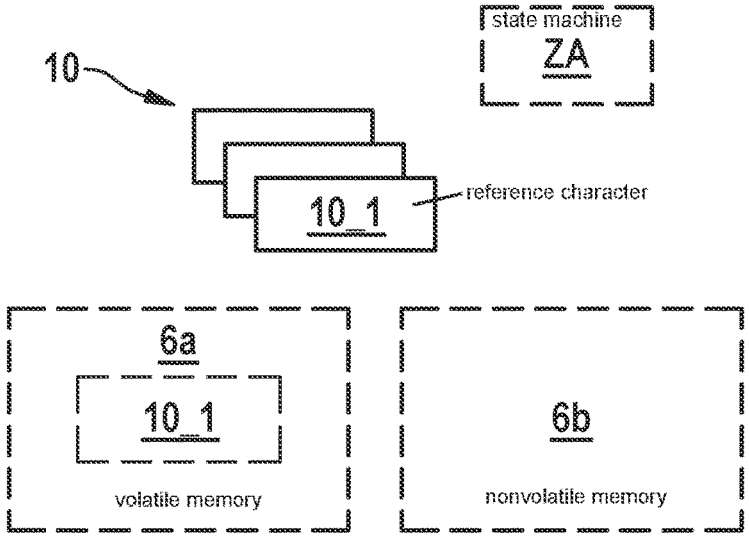

state machine
ZA
10
10_1
reference character
6a
10_1
volatile memory
6b
nonvolatile memory

Fig. 2

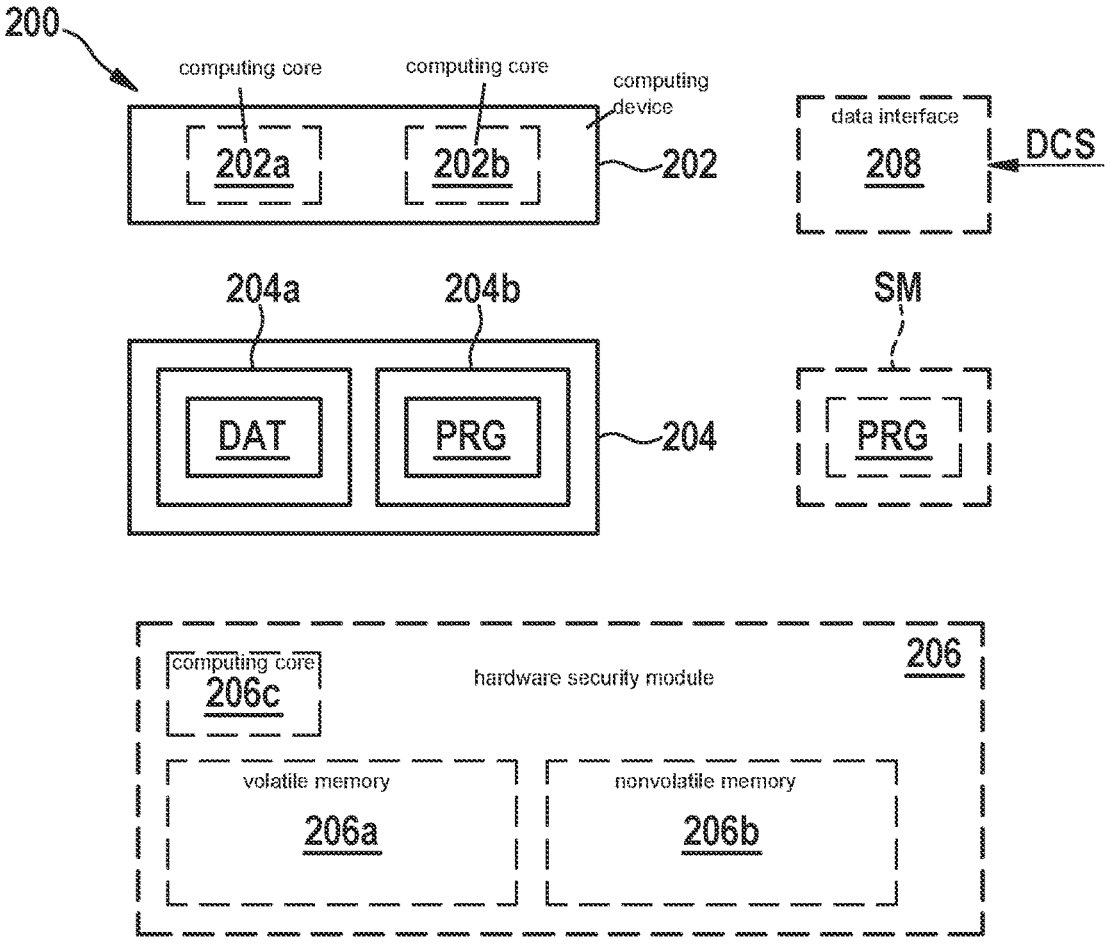

200
computing core
computing core
computing device
202a
202b
202
data interface
208
DCS
204a
204b
SM
DAT
PRG
204
PRG
computing core
206c
hardware security module
206
volatile memory
206a
nonvolatile memory
206b

Fig. 3

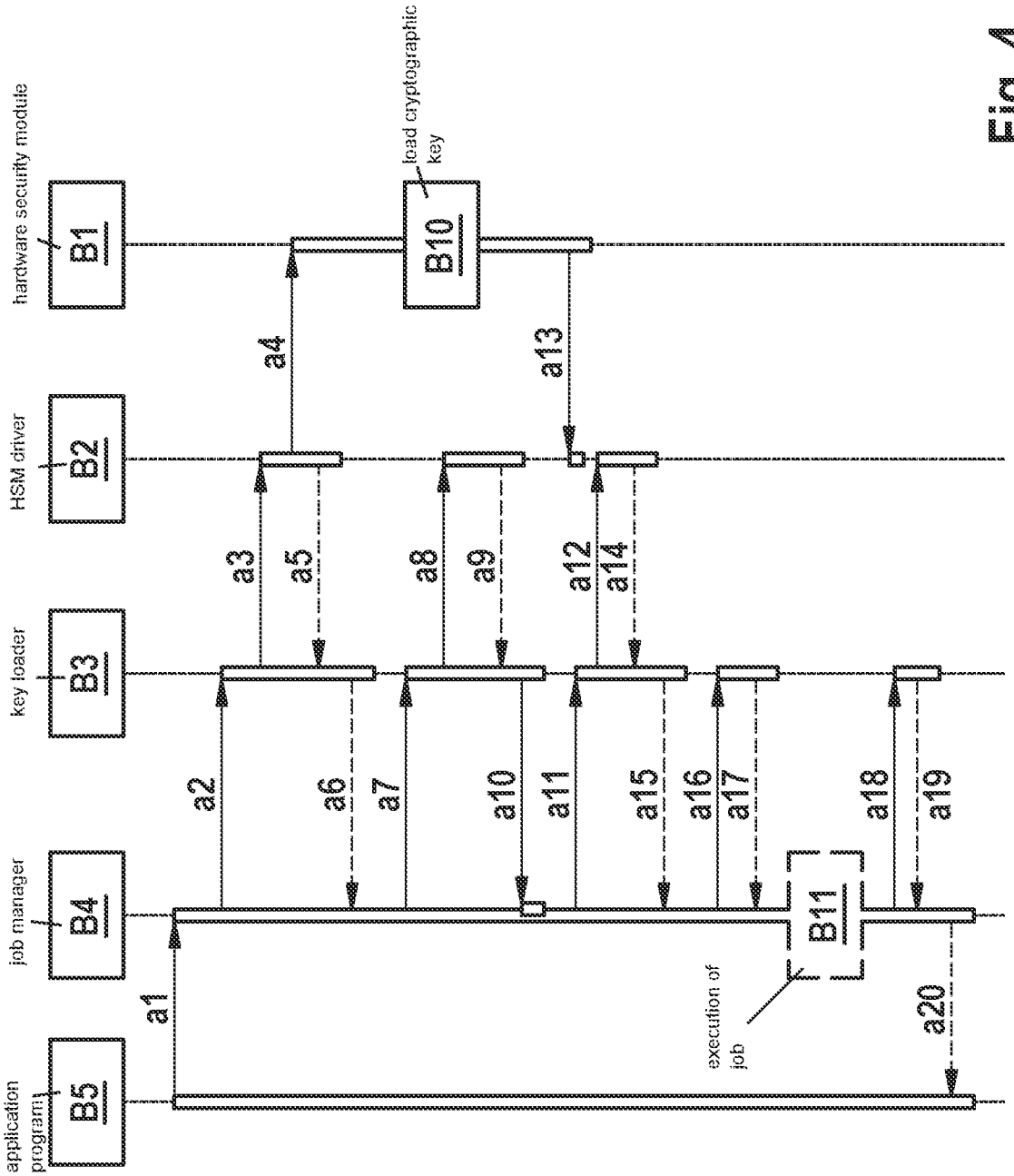

application program
B5
job manager
B4
key loader
B3
HSM driver
B2
hardware security module
B1
load cryptographic key
B10
execution of job
B11
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
a11
a12
a13
a14
a15
a16
a17
a18
a19
a20

Fig. 4

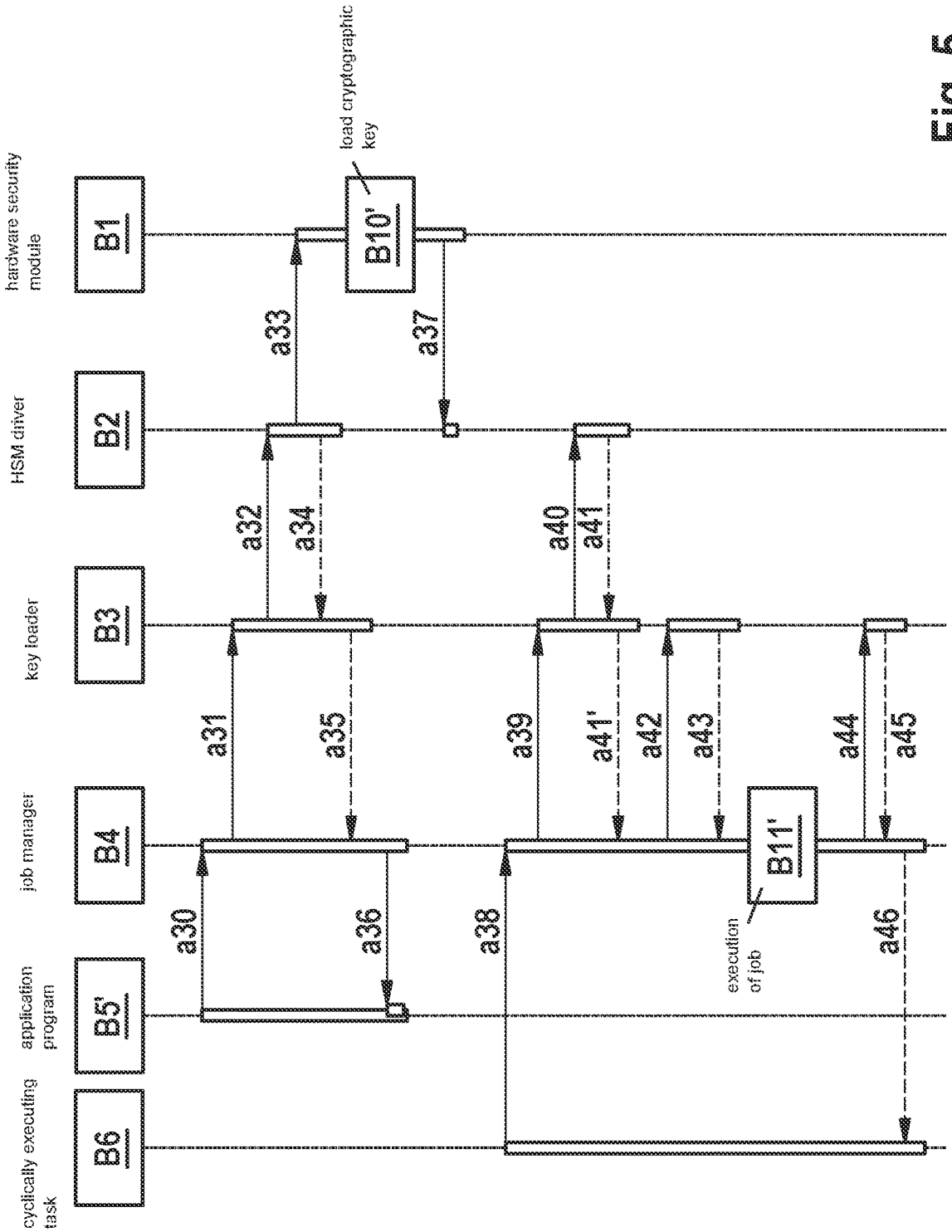

cyclically executing task
application program
job manager
key loader
HSM driver
hardware security module
B6
B5'
B4
B3
B2
B1
B10'
B11'
load cryptographic key
execution of job
a30
a31
a32
a33
a34
a35
a36
a37
a38
a39
a40
a41
a41'
a42
a43
a44
a45
a46

Fig. 5

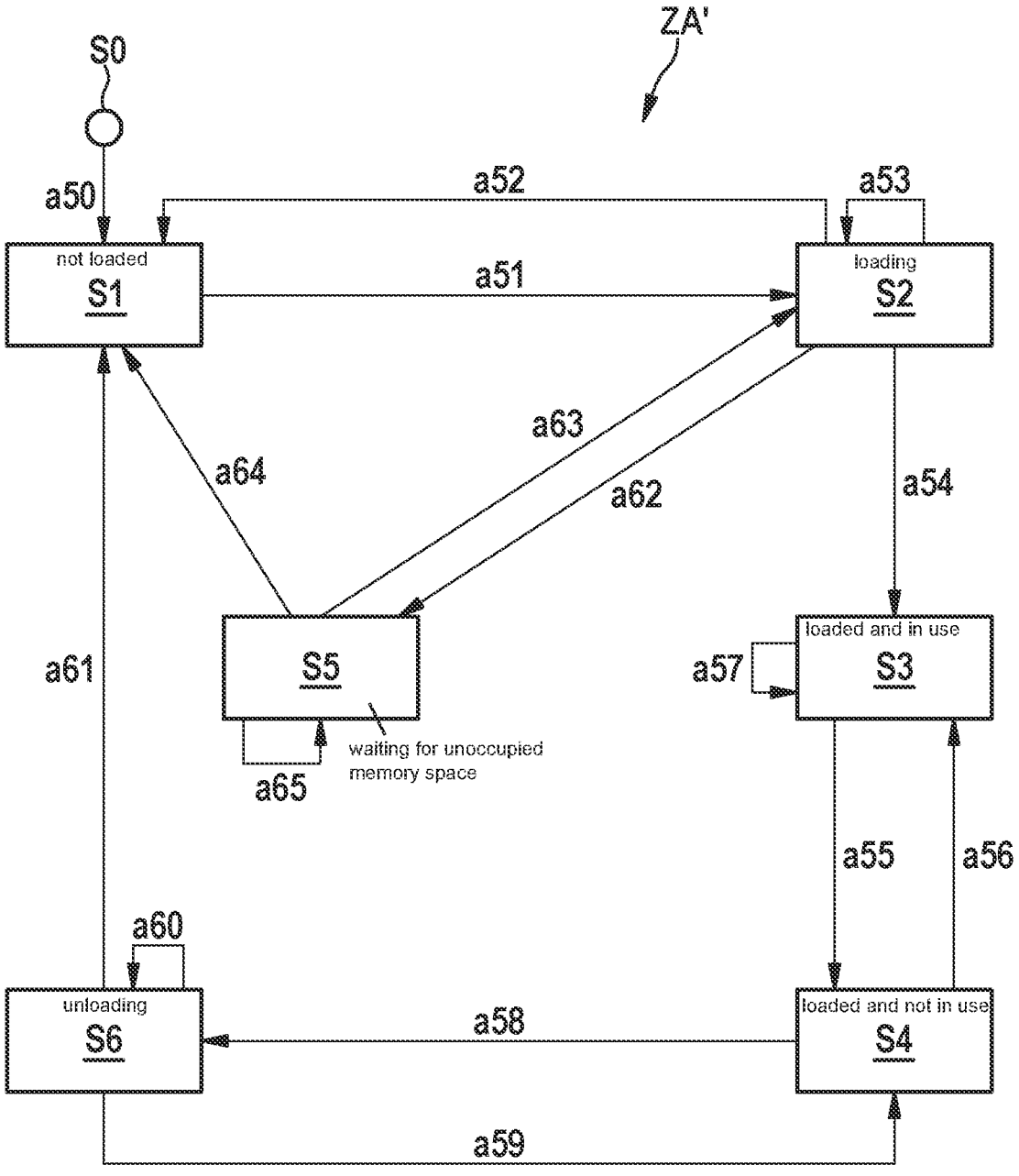

ZA'
S0
a50
a52
a53
not loaded
S1
a51
loading
S2
a63
a64
a62
a54
a61
S5
a57
loaded and in use
S3
waiting for unoccupied
memory space
a65
a55
a56
a60
unloading
S6
a58
loaded and not in use
S4
a59

Fig. 6

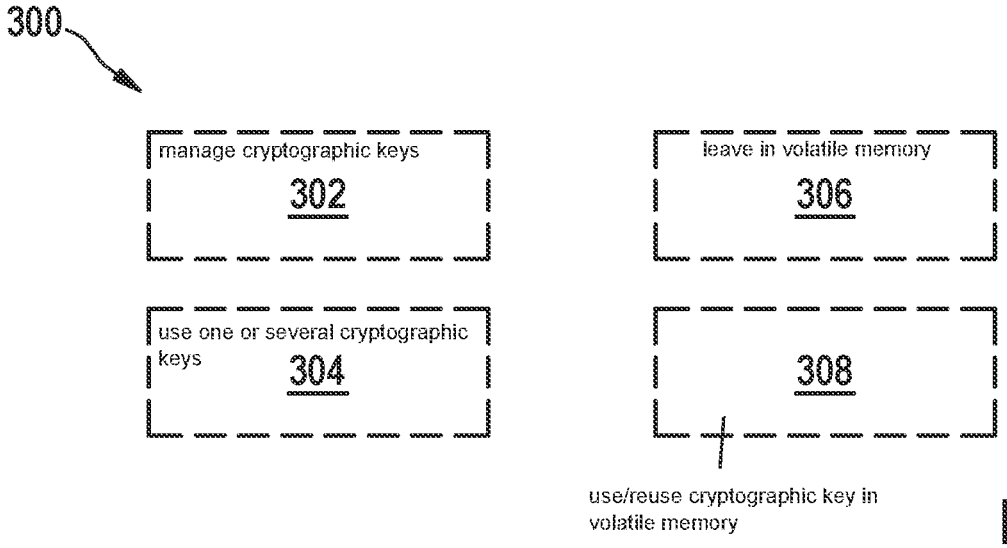

300
manage cryptographic keys
302
use one or several cryptographic keys
304
leave in volatile memory
306
308
use/reuse cryptographic key in volatile memory

Fig. 7

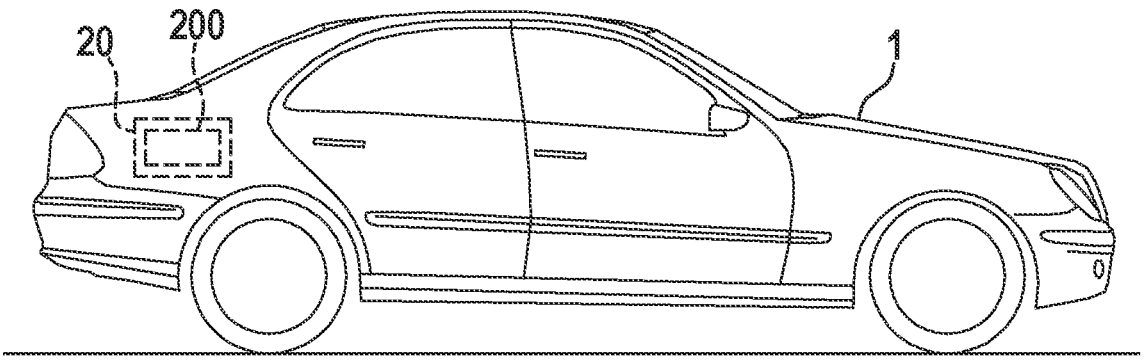

20
200
1

Fig. 8

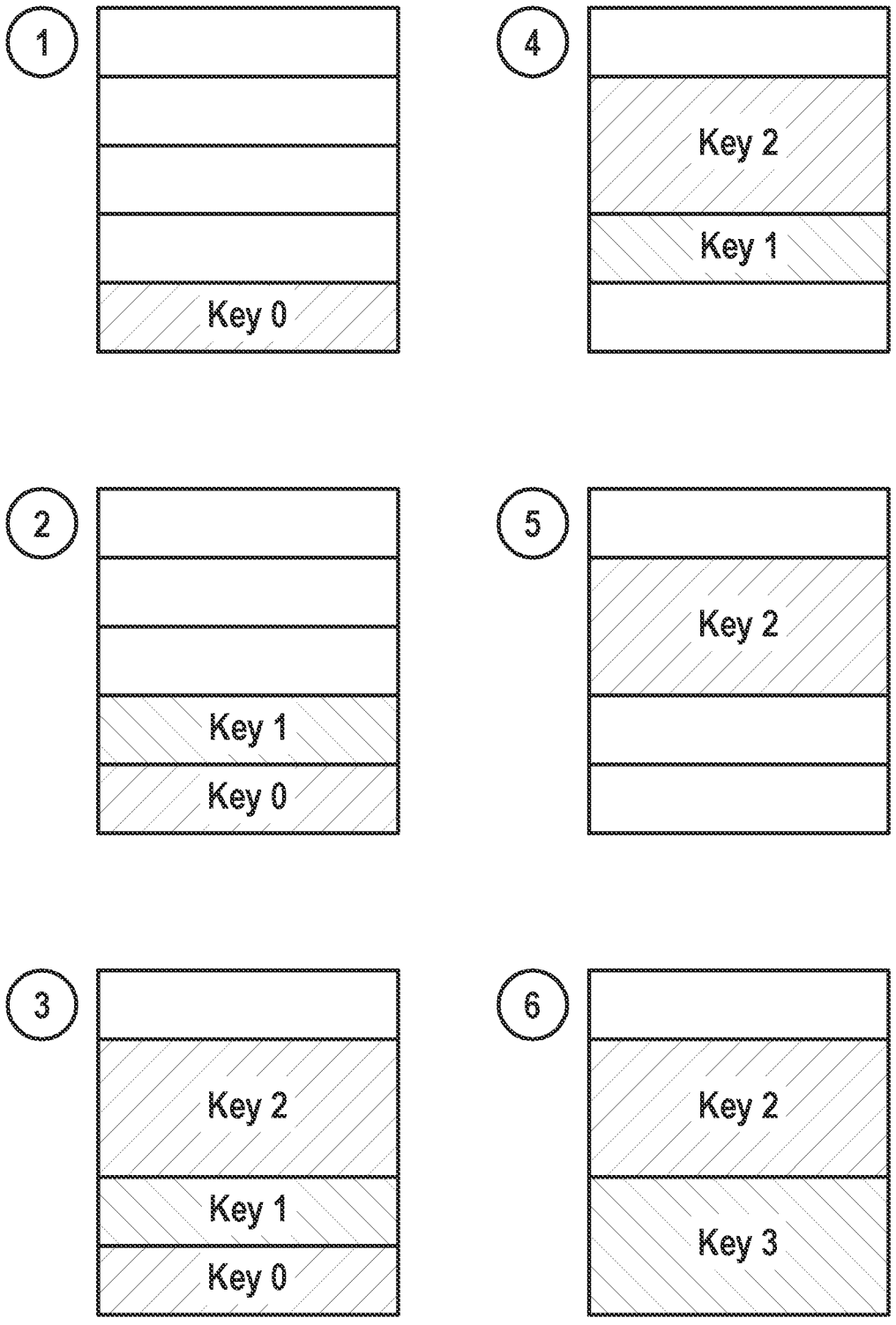

1
Key 0
4
Key 2
Key 1
2
Key 1
Key 0
5
Key 2
3
Key 2
Key 1
Key 0
6
Key 2
Key 3

Fig. 9

METHOD AND APPARATUS FOR MANAGING CRYPTOGRAPHIC KEYS

CROSS REFERENCE

[0001] The present application claims the benefit under 35 U.S.C. § 119 of German Patent Application No. 102020212772.7 filed on Oct. 9, 2020, which is expressly incorporated herein by reference in its entirety.

FIELD

[0002] The present invention relates to a method for managing cryptographic keys.

[0003] The present invention further relates to an apparatus for managing cryptographic keys.

SUMMARY

[0004] Exemplifying embodiments of the present invention refer to a method for managing cryptographic keys, for example for a control device, in particular for a motor vehicle, having the following steps: associating a state with at least one of the cryptographic keys; using the at least one cryptographic key based on the state. In further exemplifying embodiments this makes possible flexible utilization or use of the at least one cryptographic key.

[0005] In further exemplifying embodiments of the present invention, provision is made that the at least one cryptographic key is storable and/or stored at least temporarily in a volatile memory, for example a working memory or a register memory.

[0006] In further exemplifying embodiments of the present invention, provision is made that the at least one cryptographic key is storable and/or stored at least temporarily in a nonvolatile memory, for example a flash (EEPROM) memory.

[0007] In further exemplifying embodiments of the present invention, the at least one cryptographic key can also, for instance, be copied or shifted from the nonvolatile memory into the volatile memory, which can also be referred to, for example, as "loading."

[0008] In further exemplifying embodiments of the present invention, a deletion or release for overwriting a cryptographic key stored in the volatile memory can also be referred to, for example, as "unloading."

[0009] In further exemplifying embodiments of the present invention, the volatile memory and/or the nonvolatile memory can be associated, for instance, with at least one hardware security module (HSM).

[0010] In further exemplifying embodiments of the present invention, the volatile memory and/or the nonvolatile memory can be integrated, for instance, into the at least one hardware security module (HSM).

[0011] In further exemplifying embodiments of the present invention, the HSM can be associated, for example, with a control device or with a computing device of a control device and, for instance, can perform cryptographic functions or processes and, for instance, can manage the aforesaid cryptographic key.

[0012] In further exemplifying embodiments of the present invention, provision is made that the state of the at least one cryptographic key is characterized by at least one of the following elements: a) unloaded, the at least one cryptographic key being, for example, not located in a or the volatile memory (but instead, for example, in a or the nonvolatile memory); b) loaded and in use, the at least one cryptographic key, for example, being located in a or the volatile memory and currently being used, for example, by a computer program, for example a cryptography driver object; c) loaded and not in use, the at least one cryptographic key, for example, being located in a or the volatile memory and not currently being used, for example not (even) by a or the cryptography driver object; d) loading, a loading operation of the at least one cryptographic key, for example from a or the nonvolatile memory into a or the volatile memory, being, for example, not already complete; e) unloading, an unloading operation of the at least one cryptographic key, for example from a or the volatile memory, being, for example, not already complete; f) waiting for unoccupied memory space in a or the volatile memory (e.g., until memory space in the volatile memory has been cleared, for instance by deletion or unloading of another key); g) updating, for example upon request by another unit or higher-level (processing) layer. In further exemplifying embodiments, a cryptographic key having the "updating" state is not used. In further exemplifying embodiments, for example, it is possible instead to wait for a subsequent state in which updating of the cryptographic key is complete.

[0013] In further exemplifying embodiments of the present invention, one or several cryptography driver sessions or contexts, which can also be referred to as "cryptography driver objects," are implemented, for example, by way of a cryptography driver. In further exemplifying embodiments, the cryptography driver contexts are independent of one another and, for instance, can perform different tasks at least partly overlappingly in time, while one cryptography driver context is embodied in each case, for instance, to perform one task. In further exemplifying embodiments, a cryptographic key can be used, for instance, simultaneously by several different cryptography driver contexts.

[0014] In further exemplifying embodiments of the present invention, a quantity of cryptography driver contexts or cryptography driver objects is statically predefined, for example at a compile time (or "translation time" of the software, for instance, from a programming language into an object code), and is therefore constant during a runtime of the software. In further exemplifying embodiments, "jobs" are also associated statically with cryptography driver objects.

[0015] In further exemplifying embodiments of the present invention, cryptography driver contexts or cryptography driver objects are dynamically predefinable.

[0016] In further exemplifying embodiments of the present invention, a cryptography driver object is a software component (computer program) that corresponds, for instance, to a, for example, independent (hardware) device, for example an advanced encryption standard (AES) accelerator, to a component of a hardware security module, e.g., a computing core, or to a session of a hardware security module which is embodied, for instance, to execute various cryptography primitives.

[0017] In further exemplifying embodiments of the present invention, the cryptography driver object is executed on a computing core of a or the computing device ("host core").

[0018] In further exemplifying embodiments of the present invention, a cryptographic key having the "loaded and not in use" state is, for instance, a candidate for an exchange with a further cryptographic key that is intended to be used

currently or in the future and is therefore, for example, intended to be loaded into the volatile memory.

[0019] In further exemplifying embodiments of the present invention, provision is made that a specific cryptographic key has, at a point in time under consideration, exactly one state (and not, for example, several states).

[0020] In further exemplifying embodiments of the present invention, provision is made that a state machine is used to associate the state with at least one of the cryptographic keys.

[0021] In further exemplifying embodiments of the present invention, provision is made that the method further encompasses: receiving a request with regard to a cryptographic key, the request being characterized, for example, in that the cryptographic key is to be loaded into a or the volatile memory; loading the cryptographic key into the volatile memory.

[0022] In further exemplifying embodiments of the present invention, the request described above can be made, for instance, by a job manager. In further exemplifying embodiments, the job manager is a software component (computer program) that is executable, for example, by a computing core of a or the computing device ("host core") and/or that is embodied to manage or implement an execution of cryptographic methods or of steps of cryptographic methods. In further exemplifying embodiments, the job manager is embodied, for instance, to receive requests from an application program and/or to output to the application program a status characterizing an execution of a method or method step.

[0023] In further exemplifying embodiments of the present invention, the job manager can, at least temporarily, execute, for example, one or several steps of the method in accordance with the embodiments.

[0024] In further exemplifying embodiments of the present invention, it is also possible, for example, to provide a "key loader" that, for instance, is also embodied as a software component (computer program) that, for example, is executable by a computing core of the computing device and/or by a computing core of the HSM (for example, in some embodiments the key loader can also constitute part of the HSM), and/or that is embodied to execute, at least temporarily, for example, one or several steps of the method in accordance with the embodiments, for example loading and/or unloading or managing at least one cryptographic key, and/or managing states of at least one cryptographic key.

[0025] In further exemplifying embodiments of the present invention, the key loader can constitute part of a or the cryptography driver, for example similarly to the job manager. In further exemplifying embodiments, the key loader can constitute, as already mentioned above, part of the HSM.

[0026] In further exemplifying embodiments of the present invention, the job manager is embodied, for example, to execute synchronous and/or asynchronous "jobs," e.g., cryptographic methods or steps of cryptographic methods, or one or several steps of the method in accordance with the embodiments.

[0027] In further preferred embodiments of the present invention, the job manager is embodied to request, for example, the key loader to load a cryptographic key, for instance if the cryptographic key is not already located in the volatile memory.

[0028] In further exemplifying embodiments of the present invention, the key loader is embodied to load the key requested by the job manager or to cause the key to be loaded, for instance from the HSM.

[0029] In further exemplifying embodiments of the present invention, the key loader is embodied, for instance, to inform the job manager that the requested key has been successfully loaded, or that the requested key, for instance, is already contained (i.e., loaded) in the volatile memory.

[0030] In further exemplifying embodiments of the present invention, the job manager can request a reference ("handle") to the key in the context of the key loader, which reference, for instance, characterizes the copy of the corresponding key in the volatile memory or a memory location of the copy of the corresponding key in the volatile memory.

[0031] In further exemplifying embodiments of the present invention, the key loader is embodied in such a way that it asynchronously executes or initiates the loading of one or several cryptographic keys, in which context a status of the respective loading operation can be queried, for instance, from various calling contexts.

[0032] In further exemplifying embodiments of the present invention, the method further encompasses: ascertaining a first information item that characterizes whether a specific cryptographic key is a key to be stored in exclusively volatile fashion; and, optionally, executing a use of the specific cryptographic key based on the first information item, in particular refraining from nonvolatile storage of the specific cryptographic key. In further exemplifying embodiments, security can thereby be enhanced.

[0033] In further exemplifying embodiments of the present invention, the method further encompasses: ascertaining a first variable that characterizes an unoccupied memory area in the volatile memory; and, optionally, clearing or releasing memory in the volatile memory, for example in such a way that at least one further cryptographic key can be loaded into the volatile memory; the clearing being executed, for example, based on at least one predefinable algorithm; and, for example, the at least one algorithm being selectable, for example, during and/or before execution of the method.

[0034] In further exemplifying embodiments of the present invention, the clearing or release encompasses: ascertaining at least one cryptographic key that is stored in the volatile memory and is not currently being used; and releasing a memory area, occupied by the cryptographic key currently not being used, of the volatile memory, for example for overwriting with at least one further cryptographic key. In further preferred embodiments, only those cryptographic keys which, for example, are (also) stored in nonvolatile fashion are overwritten.

[0035] In further exemplifying embodiments of the present invention, the method further encompasses: leaving the at least one cryptographic key in a or the volatile memory, for example for at least one predefinable time period. In further preferred embodiments, the cryptographic key can thereby be accessed particularly quickly, for instance if it is to be used again after a first utilization. In further preferred embodiments, it is thereby possible in particular to avoid comparatively slow loading operations that involve, for instance, copying the cryptographic key from a nonvolatile memory into the volatile memory.

[0036] Further exemplifying embodiments of the present invention refer to an apparatus for executing the method in

accordance with the embodiments. The apparatus can be embodied, for example, as a computing device and/or as a hardware security module (HSM).

[0037] Further exemplifying embodiments of the present invention refer to a computer-readable storage medium encompassing instructions that, upon execution by a computer, cause the latter to execute the method in accordance with the embodiments.

[0038] Further exemplifying embodiments of the present invention refer to a computer program encompassing instructions that, upon execution by a computer, cause the latter to execute the method in accordance with the embodiments.

[0039] Further exemplifying embodiments of the present invention refer to a data carrier signal that transfers and/or characterizes the computer program in accordance with the embodiments.

[0040] Further exemplifying embodiments of the present invention refer to a use of the method in accordance with the embodiments and/or of the apparatus in accordance with the embodiments and/or of the computer-readable storage medium in accordance with the embodiments and/or of the computer program in accordance with the embodiments and/or of the data carrier signal in accordance with the embodiments for at least one of the following elements: a) managing one or several cryptographic keys, for example for a control device, in particular for a motor vehicle; b) utilizing one or several cryptographic keys, for example for a control device, in particular for a motor vehicle, for example based on a state of at least one cryptographic key and/or based on a state of at least one volatile memory; c) leaving at least one cryptographic key in a or the volatile memory; d) using, for example reusing, at least one cryptographic key that is stored in a or the volatile memory.

[0041] Further features, potential applications, and advantages of the present invention are evident from the description below of further exemplifying embodiments that are depicted in the Figures. All features described or depicted in that context, individually or in any combination, constitute the subject matter of exemplifying embodiments, regardless of their respective presentation or depiction in the description or in the figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0042] FIG. **1**A schematically shows a simplified flow chart in accordance with exemplifying embodiments of the present invention.

[0043] FIG. **1**B schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0044] FIG. **1**C schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0045] FIG. **1**D schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0046] FIG. **1**E schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0047] FIG. **2** schematically shows a simplified block diagram in accordance with further exemplifying embodiments of the present invention.

[0048] FIG. **3** schematically shows a simplified block diagram in accordance with further exemplifying embodiments of the present invention.

[0049] FIG. **4** schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0050] FIG. **5** schematically shows a simplified flow chart in accordance with further exemplifying embodiments of the present invention.

[0051] FIG. **6** schematically shows a simplified state diagram in accordance with further exemplifying embodiments of the present invention.

[0052] FIG. **7** schematically shows aspects of uses in accordance with further exemplifying embodiments of the present invention.

[0053] FIG. **8** schematically shows a control device in a destination system in accordance with further exemplifying embodiments of the present invention.

[0054] FIG. **9** schematically shows aspects in accordance with further exemplifying embodiments of the present invention.

DETAILED DESCRIPTION OF EXAMPLE EMBODIMENTS

[0055] FIG. **1**A schematically shows a simplified flow chart of a method for managing cryptographic keys in accordance with exemplifying embodiments. The method can be used, for example, for a control device **20** (see FIG. **8**), in particular for a motor vehicle **1**, but in further exemplifying embodiments is not limited thereto. The method has the following steps (see FIG. **1**A): associating **100** a state Z_**1** with at least one of the cryptographic keys; using **102** the at least one cryptographic key based on state Z_**1**. In further exemplifying embodiments, flexible use or utilization of the at least one cryptographic key is thereby made possible.

[0056] FIG. **2** shows, by way of example, several cryptographic keys **10**, one of which is labeled with the reference character **10_1**. Cryptographic keys **10** can be managed, for example, by way of the method in accordance with FIG. **1**A.

[0057] In further exemplifying embodiments, provision is made that the at least one cryptographic key **10_1** is storable and/or stored at least temporarily in a volatile memory **6***a*, for example in a working memory or a register memory.

[0058] In further exemplifying embodiments, provision is made that the at least one cryptographic key **10_1** is storable and/or stored at least temporarily in a nonvolatile memory **6***b*, for example in a flash (EEPROM) memory.

[0059] In further exemplifying embodiments, the at least one cryptographic key **10_1** can also, for instance, be copied or shifted from nonvolatile memory **6***b* into volatile memory **6***a*, which can also be referred to, for example, as "loading."

[0060] In further exemplifying embodiments, a deletion or release for overwriting a cryptographic key **10_1** stored in volatile memory **6***a* can also be referred to, for example, as "unloading."

[0061] In further exemplifying embodiments, an optional state machine ZA can be provided in order to influence, in particular to control, the state of one or several keys **10**.

[0062] Further exemplifying embodiments (see FIG. **3**) refer to an apparatus **200** for executing the method in accordance with the embodiments. Apparatus **200** has a computing device ("computer") **202** having at least one computing core **202***a*, and has a storage device **204**, asso-

ciated with computing device **202**, for at least temporary storage of at least one of the following elements: a) data DAT; b) computer program PRG, in particular for executing a method in accordance with the embodiments.

[0063] In further preferred embodiments, storage device **204** has a volatile memory **204***a* (e.g., working memory (RAM)) and/or a nonvolatile memory **204***b* (e.g., flash EEPROM).

[0064] In further preferred embodiments, computing device **202** has at least one of the following elements or is embodied as at least one of said elements: microprocessor (μP); microcontroller (μC); application-specific integrated circuit (ASIC); system on chip (SoC); programmable logic module (e.g., field programmable gate array (FPGA)); hardware circuit; or any combinations thereof.

[0065] Further preferred embodiments refer to a computer-readable storage medium SM encompassing instructions PRG that, upon execution by a computer **202**, cause the latter to execute the method in accordance with the embodiments.

[0066] Further preferred embodiments refer to a computer program PRG encompassing instructions that, upon execution of the program by a computer **202**, cause the latter to execute the method in accordance with the embodiments.

[0067] Further preferred embodiments refer to a data carrier signal DCS that characterizes and/or transfers the computer program PRG in accordance with the embodiments. Data carrier signal DCS is, for example, receivable via an optional data interface **208** of apparatus **200**.

[0068] In further exemplifying embodiments, apparatus **200** itself can be embodied as a hardware security module (HSM), a computing core **202***b* being, for instance, embodied as an HSM core.

[0069] In further exemplifying embodiments, apparatus **200** can also, for instance, have an optional hardware security module **206** that, for instance, can have a computing core **206***c* as well as, for instance, a volatile memory **206***a*, for instance similar to volatile memory **6***a* in accordance with FIG. **2**, and, for instance, a nonvolatile memory **206***b*, for instance similar to nonvolatile memory **6***b* in accordance with FIG. **2**.

[0070] In other words, in further preferred embodiments volatile memory **6***a* (FIG. **2**) and/or nonvolatile memory **6***b* can, for instance, be associated with at least one hardware security module **206** (FIG. **3**) and/or can be integrated into hardware security module **206**.

[0071] In further exemplifying embodiments, an optional computing core **202***b* ("HSM core") of computing device **200** can execute or have the function of HSM **206** or of computing core **206***c* of HSM **206**. In this case, in further preferred embodiments, for instance, the optional separate HSM **206** or its core **206***c* can be omitted.

[0072] In further exemplifying embodiments, apparatus **200** and/or HSM **206** can, for example, be associated with control device **20** (FIG. **8**) or, in the case of HSM **206**, with a computing device **202** of control device **20** and, for instance, can perform cryptographic functions or methods and, for instance, can manage the aforesaid cryptographic keys **10** (FIG. **3**).

[0073] In further exemplifying embodiments, computing device **202** (FIG. **3**) of apparatus **200** can also have one or several computing cores **202***a* for executing application programs ("host cores"), which use, for instance, cryptographic functions. In further preferred embodiments, computing device **202** can have at least one computing core **202***b* that is, for instance, physically separate from host cores **202***a* ("HSM core"), which is embodied, for instance, for the execution of cryptographic functions (for instance, creating message authentication codes (MACs), ascertaining and/or checking signatures and/or hash values), for example for the host core or cores **202***a*. In further preferred embodiments, HSM core **202***b* can, for instance, also execute management of cryptographic keys in accordance with the embodiments.

[0074] In further exemplifying embodiments, host cores **202***a* and HSM core **202***b* can, for instance, communicate directly with one another, for instance by message exchange ("message passing") and/or by way of shared memory areas, for instance of RAM **204***a*.

[0075] In further preferred embodiments it is possible to ensure, for instance by way of hardware-based security mechanisms, that host cores **202***a*, for instance, cannot access cryptographic keys **10** (FIG. **2**) that are managed by HSM core **202***b*, or corresponding memory areas in which those cryptographic keys **10** are stored.

[0076] In further exemplifying embodiments, at least one of the programs below can be executed by apparatus **200**:

[0077] 1. Host application PRG (see also, for instance, block B**5** of FIG. **4**), i.e., for instance, an application that, for instance, is executable by host core or cores **202***a* and that uses the cryptographic functions that can be furnished by HSM **206** or by HSM core **202***b;*

[0078] 2. HSM driver (see also, for instance, block B**2** of FIG. **4**), a software component or computer program that is executable by host core or cores **202***a* and constitutes an interface for the cryptographic functions of HSM **206** or of HSM core **202***b*. In further preferred embodiments, for example, the host application can communicate or interact, in particular exclusively, via the HSM driver with HSM **206** or with HSM core **202***b;*

[0079] 3. HSM software; this is, for instance, a computer program that is executable on HSM core **206***c*, **202***b* (i.e., for instance, in an environment secured with respect to host cores **202***a*) and that, for instance, executes the cryptographic functions and/or management, for instance, of cryptographic keys **10**.

[0080] In further exemplifying embodiments, provision is made that state Z_**1** (FIG. **1**) of the at least one cryptographic key **10_1** is characterized by at least one of the following elements: a) unloaded, the at least one cryptographic key **10_1** being, for example, not located in a or the volatile memory **6***a*, **206***a* (but instead, for example, in a or the nonvolatile memory **6***b*, **206***b*); b) loaded and in use, the at least one cryptographic key **10_1**, for example, being located in volatile memory **6***a*, **206***a* and currently being used, for example, by a computer program PRG, for example a cryptography driver object; c) loaded and not in use, the at least one cryptographic key **10_1**, for example, being located in a or the volatile memory **6***a*, **206***a* and not currently being used, for example not (even) by a or the cryptography driver object; d) loading, a loading operation of the at least one cryptographic key **10_1**, for example from nonvolatile memory **6***b*, **206***b* into volatile memory **6***a*, **206***a*, being, for example, not already complete; e) unloading, an unloading operation of the at least one cryptographic key **10_1**, for example from volatile memory **6***a*, **206***a*, being, for example, not already complete; f) waiting for unoccupied memory space in volatile memory **6***a*, **206***a* (e.g., until memory space in volatile memory **6***a*, **206***a* has

been cleared, for instance by deletion or unloading of another key); g) updating, for example upon request by another unit or higher-level (processing) layer. In further exemplifying embodiments, a cryptographic key having the "updating" state is not used. In further exemplifying embodiments, for example, it is possible instead to wait for a subsequent state in which updating of the cryptographic key is complete.

[0081] In further exemplifying embodiments, a cryptographic key having the "loaded and not in use" state is, for instance, a candidate for exchange with a further cryptographic key that is to be used currently or in the future and is therefore, for example, to be loaded into volatile memory **6***a*, **206***a*.

[0082] In further exemplifying embodiments, provision is made that a specific cryptographic key has, at a point in time under consideration, exactly one state (and not, for example, several states).

[0083] In further exemplifying embodiments, provision is made that state machine ZA (FIG. **2**) is used to associate state Z_**1** (FIG. **1**) with at least one of cryptographic keys **10**.

[0084] In further exemplifying embodiments (see FIG. **1**B), provision is made that the method further encompasses: receiving **110** a request A**1** with regard to a cryptographic key **10_1**, request A**1** being characterized, for example, in that cryptographic key **10_1** is to be loaded into a or the volatile memory **6***a*, **206***a*; loading **112** cryptographic key **10_1** into volatile memory **6***a*, **206***a*.

[0085] In further exemplifying embodiments, request A**1** described above can be made, for instance, by a job manager (see, for instance, block B**4** of FIG. **4**). In further exemplifying embodiments, the job manager is a computer program that is executable, for example, by HSM **206** (FIG. **3**) or by a computing core **206***c* of HSM **206** (or by a host core **202***a*) and/or that is embodied to manage an execution of cryptographic methods or of steps of cryptographic methods. In further exemplifying embodiments, the job manager is embodied, for instance, to receive requests from an application program B**5** (FIG. **4**) and/or to output to application program B**5** a status characterizing an execution of a method or method step.

[0086] In further exemplifying embodiments, the job manager can, at least temporarily, execute one or several steps of the method in accordance with the embodiments.

[0087] In further exemplifying embodiments it is also possible, for example, to provide a key loader B**3** (see, for instance, FIG. **4**) that, for instance, is also embodied as a computer program that, for example, is executable by a host core **202***a*, and/or that is embodied to execute, at least temporarily, for example, one or several steps of the method in accordance with the embodiments, for example loading and/or unloading or managing at least one cryptographic key **10_1**.

[0088] In further exemplifying embodiments, the job manager is embodied, for example, to execute synchronous and/or asynchronous jobs, e.g., cryptographic methods or steps of cryptographic methods, or one or several steps of the method in accordance with the embodiments.

[0089] In further preferred embodiments, the job manager is embodied, for example, to instruct the key loader to load a cryptographic key or cause it to be loaded, for instance if the cryptographic key is not already located in the volatile memory.

[0090] In further exemplifying embodiments, the key loader is embodied to load the key requested by the job manager or to cause the key to be loaded.

[0091] In further exemplifying embodiments, the key loader is embodied, for instance, to inform the job manager that the requested key has been successfully loaded, or that the requested key, for instance, is already contained (i.e., loaded) in the volatile memory.

[0092] In further exemplifying embodiments, the job manager can request a reference ("handle") to the key in the context of the key loader, which reference, for instance, characterizes the copy of the corresponding key in the volatile memory.

[0093] In further exemplifying embodiments, the key loader is embodied in such a way that it asynchronously executes or initiates the loading of one or several cryptographic keys, in which context a status of the respective loading operation can be queried, for instance, from various invoking contexts.

[0094] In further exemplifying embodiments (see FIG. **1**C), the method further encompasses: ascertaining **120** a first information item I**1** that characterizes whether a specific cryptographic key **10_1** is a key to be stored in exclusively volatile fashion; and, optionally, executing **122** a use of the specific cryptographic key based on first information item I**1**, in particular, for instance, refraining from nonvolatile storage of the specific cryptographic key. In further exemplifying embodiments, security can thereby be enhanced.

[0095] In further exemplifying embodiments (see FIG. **1**D), the method further encompasses: ascertaining **130** a first variable G**1** that characterizes an unoccupied memory in volatile memory **6***a*, **206***a*; and, optionally, clearing **132** or releasing memory in volatile memory **6***a*, **206***a*, for example in such a way that at least one further cryptographic key can be loaded into volatile memory **6***a*, **206***a*.

[0096] In further exemplifying embodiments (see FIG. **1**E), the clearing or release encompasses: ascertaining **132***a* at least one cryptographic key that is stored in volatile memory **6***a*, **206***a* and is not currently being used; and releasing **132***b* a memory area, occupied by the cryptographic key currently not being used, of volatile memory **6***a*, **206***a*, for example for overwriting with at least one further cryptographic key.

[0097] In further exemplifying embodiments (see FIG. **1**A), the method further encompasses: leaving **102***a* the at least one cryptographic key **10_1** in a or the volatile memory **6***a*, **206***a*, for example for at least one predefinable time period. In further preferred embodiments, cryptographic key **10_1** can thereby be accessed particularly quickly, for instance if it is to be used again after a first utilization. In further preferred embodiments, it is thereby possible in particular to avoid comparatively slow loading operations that involve, for instance, copying cryptographic key **10_1** from a nonvolatile memory **6***b*, **206***b* into volatile memory **6***a*, **206***a*.

[0098] FIG. **4** schematically shows a simplified flow chart for, for instance, synchronous processing of jobs in accordance with further exemplifying embodiments. Block B**1** symbolizes a hardware security module (see also, for instance, reference character **206** of FIG. **3**), block B**2** symbolizes an HSM driver, block B**3** symbolizes a key loader, block B**4** symbolizes a job manager, and block B**5** symbolizes an application program.

[0099] Arrow a**1** symbolizes a request of application program B**5** to job manager B**4** to execute a job. Job manager B**4** instructs key loader B**3** (see arrow a**2**) to load a cryptographic key (usable, for example, for the job to be executed) and, if applicable, to lock it with respect to overwriting (e.g., in order to recover unoccupied memory space in the volatile memory). Key loader B**3** signals to HSM B**1** via HSM driver B**2**, by way of arrows a**3**, a**4**, that a loading operation for a cryptographic key **10_1** (FIG. **2**) is to be executed.

[0100] In further exemplifying embodiments, element a**2** (and, for instance, also element a**31** in accordance with FIG. **5**; see below) does not provide that the cryptographic key is locked in the sense that it is not usable, for instance, at least temporarily in chronologically overlapping fashion by other cryptography driver objects. Element a**2** (and, for instance, also element a**31** in accordance with FIG. **5**; see below) instead provides, for instance, that the cryptographic key is locked in the sense that it is not selectable for release, for instance if the volatile memory for the cryptographic keys is full and a release of memory space therein is therefore desirable.

[0101] In further exemplifying embodiments it is therefore possible, for instance, that, for instance, as soon as a key is loaded, a simultaneous or at least partly chronologically overlapping use of that key occurs, for instance, by different cryptography driver objects.

[0102] In further exemplifying embodiments one or several, for instance all, cryptography drivers or cryptography driver objects can set a respective lock with respect to a key (the locks are, for instance, not mutually exclusive). In further preferred embodiments this has the effect that a key, for instance once it has been loaded, remains in use as long as the total number of locks has a non-negligible value (greater than zero).

[0103] In further exemplifying embodiments, a key can transition into the "loaded and not in use" state (see FIG. **6** below) when the last lock has been removed. In further preferred embodiments, the relevant key can then, for instance, also be overwritten again, for instance in order to release memory.

[0104] Block B**10** correspondingly symbolizes the loading of cryptographic key **10_1**, for instance, from nonvolatile memory **206***b* into volatile memory **206***a* (FIG. **3**).

[0105] In further exemplifying embodiments, HSM driver B**2** signals to key loader B**3** (see arrow a**5**) that the cryptographic key is currently being loaded and the cryptographic key thus has the "loading" state. In further preferred embodiments, the situation is comparable for components B**3**, B**4** (see arrow a**6**).

[0106] In further exemplifying embodiments, job manager B**4** queries the current state of the cryptographic key (see arrows a**7**, a**8**), and components B**2**, B**3** correspondingly report back a current status (e.g., "key (still) loading") (see arrows a**9**, a**10**).

[0107] In further exemplifying embodiments, arrow a**13** symbolizes the fact that HSM B**1** signals the "key loaded" state to HSM driver B**2**. In further preferred embodiments this "key loaded" state can correspondingly be reported, after a new request all by job manager B**4** (see also request a**12** of key loader B**3**), to components B**3**, B**4** in the form of signals a**14**, a**15**.

[0108] In further exemplifying embodiments, job manager B**4** asks key loader B**3** for a reference ("handle") to the loaded cryptographic key (see arrow a**16**). In further preferred embodiments, key loader B**3** signals the reference to the loaded cryptographic key to job manager B**4** (see arrow a**17**). Block B**11** symbolizes, by way of example, execution of the job characterized by arrow a**1**, for instance using the cryptographic key that has meanwhile been loaded.

[0109] In further exemplifying embodiments, the cryptographic key is unloaded or released again (see arrow a**18**) after execution B**11** of the job; for instance, a memory area of the volatile memory which since then has been occupied by the cryptographic key can also be released and is thus usable, if applicable, for another key. Optionally, a lock that may have been applied to the key (see arrow a**2**) can also be canceled. By way of signal a**18**, job manager B**4** can signal to key loader B**3**, for example, that the relevant key is not being used at present. If, for example, the relevant key is not being used by any of the, if applicable, several HSM driver objects, the relevant key can assume, for instance, the "loaded and not in use" state.

[0110] Arrow a**19** symbolizes an optional confirmation of release a**18** on the part of key loader B**3**.

[0111] In further exemplifying embodiments, job manager B**4** can signal to application program B**5** that the job has been executed (see arrow a**20**).

[0112] In further exemplifying embodiments, job manager B**4** can check, for instance after receiving inquiry a**1**, whether a cryptographic key is necessary or indicated for execution of the job. In further preferred embodiments, for example, cryptographic services or functions that do not require a cryptographic key for their execution, for instance the calculation of hash values, can also be provided and/or used. If a cryptographic key is necessary or indicated for execution of the job, job manager B**4** requests loading from key loader B**3** (see arrow a**2**).

[0113] In further exemplifying embodiments, key loader B**3** can manage the state of at least one cryptographic key **10_1** (FIG. **2**), for example of several or all cryptographic keys **10**. For example, after receiving request a**2**, key loader B**3** can ascertain, in accordance with FIG. **4**, that the cryptographic key to be used for execution of the job is not already present in the volatile memory, and can therefore initiate the loading operation (see arrow a**3**), the loading operation being executed, for instance, asynchronously by HSM B**1** (see also, for instance, HSM cores **206***c*, **202***b*).

[0114] In further exemplifying embodiments, job manager B**4** can query the state of the loading operation (see, for instance, arrows a**7**, all), for instance until the key is loaded.

[0115] FIG. **5** schematically shows a simplified flow chart for, for instance, asynchronous processing of jobs in accordance with further exemplifying embodiments. Block B**1** symbolizes a hardware security module (see also, for instance, reference character **206** of FIG. **3**), block B**2** symbolizes an HSM driver, block B**3** symbolizes a key loader, block B**4** symbolizes a job manager, block B**5'** symbolizes an application program, and block B**6** symbolizes a cyclically executed task, i.e., for instance a cyclically executed process.

[0116] Arrow a**30** symbolizes a request by application program B**5'** to job manager B**4** to execute a job. Job manager B**4** instructs key loader B**3** (see arrow a**31**) to load (or cause the loading of) a cryptographic key (for example, one usable for the job to be executed) and, if applicable, to lock it, for example in order to protect it from overwriting. Key loader B**3** signals to HSM B**1**, via HSM driver B**2** by

way of arrows a**32**, a**33**, that a loading operation for a cryptographic key **10_1** (FIG. **2**) is to be executed.

[0117] Block B**10'** correspondingly symbolizes the loading of cryptographic key **10_1**, for instance, from nonvolatile memory **206***b* into volatile memory **206***a* (FIG. **3**).

[0118] Arrows a**34**, a**35** symbolize signaling to components B**3**, B**4** with regard to the current state of the loading of the cryptographic key, and arrow a**36** comparably symbolizes signaling of the status of execution of the job to application program B**5'**.

[0119] In further exemplifying embodiments, HSM B**1** signals a return code regarding loading B**10'** (see arrow a**37**) to HSM driver B**2**.

[0120] In further exemplifying embodiments, task B**6** signals the execution of a planned function to job manager B**4** (see arrow a**38**), which then queries the loading state of the previously requested cryptographic key (see arrows a**39**, a**40**). HSM driver B**2** signals that the cryptographic key has now been loaded (see arrows a**41**, a**41'**).

[0121] In further exemplifying embodiments, job manager B**4** asks key loader B**3** for a reference ("handle") to the loaded cryptographic key (see arrow a**42**). In further preferred embodiments, key loader B**3** signals the reference to the loaded cryptographic key to job manager B**4** (see arrow a**43**). Block B**11'** symbolizes, by way of example, execution of the job characterized by arrow a**30**, for instance using the cryptographic key that has meanwhile been loaded.

[0122] In further exemplifying embodiments, the cryptographic key is unloaded or released again (see arrow a**44**) after execution B**11'** of the job; a memory area of the volatile memory which since then has been occupied by the cryptographic key can also, for instance, be released and is thus usable, if applicable, for another key. Arrow a**45** symbolizes an optional confirmation of release a**44** on the part of key loader B**3**. Arrow a**46** symbolizes, by way of example, completion of the planned function.

[0123] In further exemplifying embodiments, in the context of the exemplifying asynchronous execution of the task in accordance with FIG. **5**, the latter can be started by way of request a**30** while a further execution is being performed, for instance by way of cyclic task B**6**. In further preferred embodiments, the principle of loading a cryptographic key can correspond to the steps described by way of example with reference to FIG. **4**.

[0124] In further exemplifying embodiments, computing time on a host core **202***a* (FIG. **3**) that, for instance, is executing application program B**5**, B**5'**, which time host core **202***a* would otherwise need to use, for instance, for a blocking polling loop, for instance in order itself to wait for completion of the loading of the cryptographic key, can be saved by the procedure described by way of example above.

[0125] In further exemplifying embodiments, a throughput of host core **202***a*, for instance in an automotive open system architecture (AUTOSAR) system, can therefore be increased by way of the procedure described by way of example above.

[0126] Utilization of the management of cryptographic keys in accordance with the embodiments makes possible, at least temporarily and/or at least in some embodiments, a reduction in latency in the execution of jobs, for instance by the fact that at least some keys can be left in the comparatively fast volatile memory even when a job that has used the key or keys has been completed, for example in contradistinction to constant deletion of the keys from the volatile memory directly after use, and constant reloading of the keys. This means, for instance, that a subsequent job that is intended to use the key or keys finds the key or keys, for instance, already in the volatile memory (provided it has not, for instance, already been overwritten in accordance with further exemplifying embodiments), so that the key or keys does/do not first need to be loaded into the volatile memory.

[0127] In further exemplifying embodiments, various methods for exchanging or overwriting keys can be provided, which are embodied to retain or leave in the volatile memory those keys that will be (re)used in the future with the highest probability and/or, for instance, the keys having a highest priority; this offers a performance advantage and increases the throughput (characterizable, for instance, by a number of jobs executed in a predefinable time) of a system.

[0128] In further exemplifying embodiments, several jobs can be executed simultaneously or at least in part in chronologically overlapping fashion, for example by HSM driver B**2** (FIGS. **4**, **5**).

[0129] While the exemplifying embodiments in accordance with FIGS. **4**, **5** describe possible scenarios in which a cryptographic key is loaded, for instance, from a nonvolatile memory into a volatile memory, in further exemplifying embodiments one or several of the situations recited below, inter alia, can also occur.

[0130] In further exemplifying embodiments, a cryptographic key can already be present in the volatile memory, for instance because it has already been loaded earlier. In further preferred embodiments, a cryptographic key can be embodied, for instance, as a "startup" key that is loaded into the volatile memory upon startup of HSM **206**.

[0131] In further exemplifying embodiments, a cryptographic key can be embodied, for instance, as a key to be stored (for instance, only) in volatile fashion, and can thus, for instance, already be present (for instance, only) in the volatile memory.

[0132] In further exemplifying embodiments, the unoccupied memory space in the volatile memory may no longer be sufficient for loading of a (for instance, further) cryptographic key, in which context, if applicable, at least one of the cryptographic keys located in the volatile memory can be unloaded or released. In further preferred embodiments, provision can be made that one or several of the cryptographic keys present in the volatile memory must not be unloaded; this can be, for instance, a key to be stored (for instance, only) in volatile fashion, and/or a "startup" key, and/or a cryptographic key that is currently being used by another HSM driver object.

[0133] In further exemplifying embodiments, key loader B**3** can furnish or execute one or several of the following aspects:

[0134] a) loading a cryptographic key stored in the nonvolatile memory into the volatile memory, for instance when job manager B**4** requests it (see, for instance, arrow a**2** of FIG. **4**);

[0135] b) ascertaining which keys are keys to be stored in volatile fashion, in particular in order to prevent them from being unloaded;

[0136] c) releasing volatile memory so that a new key can be loaded, in which context, for example, no key currently being used is unloaded;

[0137] d) ascertaining which keys are "startup" keys, in particular in order to prevent them from being unloaded;

[0138] e) ascertaining which keys are being used, for instance, by each of the possibly several HSM driver objects.

[0139] FIG. **6** is a schematic simplified state diagram in accordance with further exemplifying embodiments which illustrates some possible states of a cryptographic key **10_1** (FIG. **2**) in accordance with further exemplifying embodiments. The states that are depicted by way of example can correspond, for example, to a subset of states controllable by way of a state machine ZA' (see also state machine ZA for controlling states of cryptographic key **10** in accordance with FIG. **2**). For example, state machine ZA in accordance with FIG. **2** can have the configuration ZA' in accordance with FIG. **6**.

[0140] Block S**0** symbolizes an initial state from which a change is made, by way of state transition a**50**, to the "unloaded" or "not loaded" state S**1**.

[0141] From state S**1**, "not loaded," in further preferred embodiments a change can be made by way of state transition a**51** to state S**2**, "loading." This can be the case, for instance, when key loader B**3** (FIG. **4**) requests from HSM driver B**2** the loading of a cryptographic key (see arrow a**3** of FIG. **4**).

[0142] In further preferred embodiments, a state transition a**52** characterizes a change from state S**2** back to state S**1**, for instance if loading has not been successfully executed. In further preferred embodiments, this state transition a**52** can also occur if it is not possible to find a key that can be unloaded.

[0143] In further preferred embodiments, a state transition a**53** symbolizes the retention of state S**2**, thus characterizing, for instance, the fact that loading of the cryptographic key is continuing.

[0144] In further preferred embodiments, a state transition a**54** characterizes a change from state S**2** to state S**3**, "in use" or "loaded and in use."

[0145] In further preferred embodiments, state transition a**55** characterizes a change from state S**3** to state S**4**, "loaded and not in use."

[0146] In further preferred embodiments, a state transition a**56** characterizes a change from state S**4** to state S**3**, in which the relevant cryptographic key can be locked, for instance so that it cannot be selected for release, for instance if the volatile memory for the cryptographic keys is full and a release of memory space therein is desirable.

[0147] In further preferred embodiments, a state transition a**57** symbolizes the retention of state S**3**, thus characterizing, for instance, the fact that the cryptographic key is still "loaded and in use."

[0148] In further preferred embodiments, a state transition a**58** characterizes a change from state S**4** to state S**6**, "unloading," in which the relevant cryptographic key or the memory area of the, for instance, volatile memory used by it is released.

[0149] In further preferred embodiments, a state transition a**59** characterizes a change from state S**6** back to state S**4**, for example if release of the key or of the associated memory area cannot be executed successfully.

[0150] In further preferred embodiments, a state transition a**60** symbolizes the retention of state S**6**, thus characterizing, for instance, the fact that unloading or release is continuing.

[0151] In further preferred embodiments, a state transition a**61** characterizes a change from state S**6** to state S**1**, for instance when release of the key or of the associated memory area has been completed.

[0152] In further preferred embodiments, a state transition a**62** characterizes a change from state S**2** to state S**5**, "waiting for unoccupied memory space, for instance, in the volatile memory," for instance if a unoccupied memory area for the key to be loaded first needs to be created for loading from state S**2**.

[0153] A state transition a**65** from state S**5** to state S**5** itself symbolizes the retention of state S**5** in accordance with further exemplifying embodiments, for instance when the release of a memory area for acceptance of a new key to be loaded is continuing.

[0154] In further exemplifying embodiments, a state transition a**63** characterizes a change from state S**5** to state S**2**, for instance when unoccupied memory space for the key to be loaded has been created for loading in accordance with state S**2**.

[0155] In further preferred embodiments, further states in addition to those illustrated here by way of example in FIG. **6** can be provided and, for instance, can be associated at least temporarily with at least one of cryptographic keys **10** (FIG. **2**).

[0156] In further exemplifying embodiments, provision is made that state machine ZA (FIG. **2**) is used in order to associate the respective state S**1**, S**2**, etc. with at least one of cryptographic keys **10**. By way of example, the state labeled by way of example with the reference character Z_**1** in FIG. **1**A can correspond at least temporarily to at least one of states S**1**, S**2**, etc. illustrated here by way of example in FIG. **6**.

[0157] In further preferred embodiments, it may happen that several jobs that are being executed at least in part in chronologically overlapping fashion use the same cryptographic key. In further preferred embodiments, no conflicts arise in this context because key loader B**3** (FIG. **4**) assigns a respectively corresponding state, e.g., "loading," to the key that is to be loaded.

[0158] In further preferred embodiments, it is possible to provide a selection method that can be used to ascertain which of, if applicable, several unused loaded cryptographic keys is to be unloaded or to be unloaded first, for instance when unoccupied memory is required in volatile memory **6***a*, **206***a* for loading new keys.

[0159] In further preferred embodiments, the selection method can be embodied configurably, for instance configurably in a pre-compile phase.

[0160] In further preferred embodiments, the selection method can be based on information that is available, for instance, at a runtime of apparatus **200**, **206**, for instance:

[0161] a) unloading the key that, of all the loaded keys, has not been used for the longest time;

[0162] b) unloading the key that has been used least often;

[0163] c) unloading a key selected on a random or pseudo-random basis.

[0164] In further preferred embodiments, the selection method can be based on an individual priority of the cryptographic keys, which is assigned, for instance, statically to the keys. In further preferred embodiments, for instance, the key that is currently unused, having the lowest priority, can correspondingly be unloaded.

[0165] Further exemplifying embodiments (see FIG. **7**) refer to a use 300 of the method in accordance with the embodiments and/or of the apparatus in accordance with the

embodiments and/or of the computer-readable memory medium in accordance with the embodiments and/or of the computer program in accordance with the embodiments and/or of the data carrier signal in accordance with the embodiments for at least one of the following elements: a) managing **302** one or several cryptographic keys **10**, **10_1**, for example for a control device **20**, in particular for a motor vehicle **1**; b) using **304** one or several cryptographic keys, for example for a control device, in particular for a motor vehicle, for example based on a state Z_1 of at least one cryptographic key and/or based on a state of at least one volatile memory **6***a*, **206***a*; c) leaving **306** at least one cryptographic key in a or the volatile memory; d) using **308**, for example reusing, at least one cryptographic key that is stored in a or the volatile memory.

[0166] FIG. **9** schematically shows aspects in accordance with further exemplifying embodiments which refer, for instance, to an unloading of keys from the volatile memory or a replacement or overwriting.

[0167] If the memory for the keys, for instance volatile memory **6***a*, **206***a*, is full, for instance does not have sufficient unoccupied memory space for loading or storing a new key, then in accordance with further exemplifying embodiments one or several keys present in memory **6***a*, **206***a* are unloaded. In further preferred embodiments, the key loader ascertains or indicates which key or keys is/are to be unloaded, for instance based on a selected algorithm.

[0168] In further preferred embodiments, the key loader can, for instance, unload successive keys, for instance, until sufficient unoccupied memory is (again) present in volatile memory **6***a*, **206***a*, for instance in order to load at least one new key or to store it at least temporarily in volatile memory **6***a*, **206***a*.

[0169] FIG. **9** shows, by way of example, a diagram for the unloading of keys in accordance with further exemplifying embodiments; in step **1** a job 0 requires "Key 0," and "Key 0," is therefore loaded, for instance, from a nonvolatile memory into volatile memory **6***a*, **206***a*. In FIG. **9**, occupancy of the volatile memory is symbolized by the "table" having five rows and one column.

[0170] In step **2**, a job 1 requires "Key 1," and "Key 1" is therefore also loaded. In step **3**, a job 2 requires "Key 2," and "Key 2" is therefore also loaded. "Key 2" is, for instance, twice the size of the "Key 0" and "Key 1" keys.

[0171] It is assumed by way of example that proceeding from the state shown in step **3** of FIG. **9**, a further job is intended to use a further key ("Key 3") for which sufficient unoccupied memory in the volatile memory is not available. For example, the further "Key 3" has the same size as "Key 2." In further embodiments, loading of the further "Key 3" can therefore fail, and in accordance with further exemplifying embodiments the key loader can unload one or several of the loaded keys ("Key 0," "Key 1").

[0172] In further preferred embodiments, in which the selected algorithm provides, for unloading, that the key to be unloaded is the one that (with reference to the further loaded keys) has not been used for the longest time ("least recently used"), the key loader will thus, for instance, unload "Key 0" or release its memory space in the volatile memory (see step **4** of FIG. **9**), and if applicable will attempt again to load the further key ("Key 3"). In step **4**, however, sufficient unoccupied continuous memory is still not available for this. In step **5**, the key loader therefore also unloads "Key 1." In step **6**, the key loader once again loads the further key ("Key 3"), this time successfully, since sufficient continuous unoccupied memory is now available. In further preferred embodiments, the further job that is using the further key ("Key 3") can then be executed.

What is claimed is:

1. A method for managing cryptographic keys, comprising the following steps:

associating a state with at least one of the cryptographic keys;

using the at least one cryptographic key based on the state.

2. The method as recited in claim **1**, wherein the managing of the cryptographic keys is for a control device of a motor vehicle.

3. The method as recited in claim **1**, wherein: a) the at least one cryptographic key is storable and/or stored at least temporarily in a volatile memory; and/or b) the at least one cryptographic key is storable and/or stored at least temporarily in a nonvolatile memory.

4. The method as recited in claim **1**, wherein the state is characterized by at least one of the following elements: a) unloaded, the at least one cryptographic key being not located in a volatile memory; b) loaded and in use, the at least one cryptographic key being located in the volatile memory and currently being used; c) loaded and not in use, the at least one cryptographic key being located in the volatile memory and not currently being used; d) loading, a loading operation of the at least one cryptographic key from a nonvolatile memory into the volatile memory being not already complete; e) unloading, an unloading operation of the at least one cryptographic key from the volatile memory not already complete; f) waiting for unoccupied memory space in the volatile memory; g) updating.

5. The method as recited in claim **1**, wherein a state machine is used to associate the state with the at least one of the cryptographic keys.

6. The method as recited in claim **1**, further comprising:

receiving a request with regard to a cryptographic key, the request being to load the cryptographic key into a volatile memory; and

loading the cryptographic key into the volatile memory.

7. The method as recited in claim **1**, further comprising:

ascertaining a first information item that characterizes whether a specific cryptographic key is a key to be stored in exclusively volatile fashion.

8. The method as recited in claim **7**, further comprising:

executing a use of the specific cryptographic key based on the first information item including refraining from nonvolatile storage of the specific cryptographic key.

9. The method as recited in claim **1**, further comprising:

ascertaining a first variable that characterizes an unoccupied memory area in the volatile memory.

10. The method as recited in claim **9**, further comprising:

clearing memory in the volatile memory in such a way that at least one further cryptographic key can be loaded into the volatile memory, the clearing being executed based on at least one predefinable algorithm; the at least one algorithm being selectable during and/or before execution of the method.

11. The method as recited in claim **10**, wherein the clearing includes:

ascertaining at least one cryptographic key that is stored in the volatile memory and is not currently being used; and

releasing a memory area occupied by the cryptographic key currently not being used of the volatile memory for overwriting with at least one further cryptographic key.

12. The method as recited in claim **1**, further comprising:

leaving the at least one cryptographic key in a volatile memory for at least one predefinable time period.

13. An apparatus configured to manage cryptographic keys, the apparatus configured to:

associate a state with at least one of the cryptographic keys;

use the at least one cryptographic key based on the state.

14. A non-transitory computer-readable storage medium on which are stored instructions for managing cryptographic keys, the instructions, when executed by a computer, causing the computer to perform the following steps:

associating a state with at least one of the cryptographic keys;

using the at least one cryptographic key based on the state.

15. The method as recited in claim **1**, wherein the method is used for at least one of the following:

a) managing one or several cryptographic keys for a control device for a motor vehicle;

b) utilizing one or several cryptographic keys for a control device for a motor vehicle based on a state of at least one of the cryptographic keys and/or based on a state of at least one volatile memory;

c) leaving at least one cryptographic key in the volatile memory;

d) using or reusing at least one cryptographic key that is stored in the volatile memory.

* * * * *