

Secure element for securely processing digital information

TECHNICAL FIELD

[0001] The present disclosure relates to the field of computer security and
5 more precisely to securely processing digital information, such as codes
(applications) and/or data.

BACKGROUND

[0002] Generally, a processing device that can securely process digital
information, also known as a secure element, for example a smart card or a
10 chip, includes hardware resources such as one or more processors (e.g. a
host processor and a secure processor), and one or more memories (volatile
memory, cache memory and non-volatile memory). The processing device
usually operates under control of an operating system and executes program
instructions by using one or more software components or applications.
15 When an application is executed, the digital information (code and data)
required for the application and the data produced by the application can be
stored in a non-volatile memory and, when required to be executed or
processed, the digital information may be loaded in a volatile memory or
cache memory.

20 [0003] The secure element may be embedded in a SoC (System on the Chip)
having several processing modules, several memories and several
functionalities. The SoC may be integrated in a larger module, for example a
IoT device (Internet of Things device).

[0004] In a constraint environment like an IoT system, the secure element
25 may have only a small non-volatile memory, such as an OTP (one time
programmable) memory, that is used to store a limited amount of information,
typically counters and/or keys. The secure element is not used to store the
digital information (e.g. code and/or data). To keep the cost of the secure
element low, the digital information is stored in an external non-volatile

memory located outside the secure element (for example a flash memory in the SoC or in the module outside the SoC). This external memory has a large storage capacity, for example of several megabytes, higher than the internal storage capacity of the secure element. The digital information stored in the

5 external memory shall be protected with a high security level equivalent to the security level of the secure processor. More precisely, it should be secured in privacy, integrity, authenticity and freshness. Regarding the freshness, it means that the digital information should be protected against a replay attack (also known as playback attack), that is a form of network attack in which a

10 valid data transmission is maliciously or fraudulently repeated or delayed.

[0005] The secure element needs to be protected against malicious attacks, in particular against replay attacks. A replay attack (also known as playback attack) is a malicious attack in which a hacker eavesdrops on a valid data transmission, intercepts it, and then fraudulently delays or repeats it to

15 misdirect a receiver (in this case, the secure element) into doing what the hacker wants.

[0006] The secure element has a loader that securely imports the external digital information in an internal memory such as a RAM (Random-Access Memory). When a piece of digital information is imported, it shall be verified

20 whether or not it has been replaced by a previous version of said piece of digital information that is no longer valid.

[0007] EP 2 860 660 discloses a data processing system including a SoC having a host processor (HCPU) and associated host memories, a secure processor (SCPU) and associated secure memories, a secure cache memory

25 storing a plurality of cache lines and a cache controller that controls loading the cache lines from the host processor to the secure processor. The cache memory stores an integrity table containing a fingerprint of each cache line, that has been computed using a hash function. When a cache line is loaded into the secure processor, its integrity is verified by successfully comparing a

fingerprint of the cache line computed by the secure processor and the corresponding fingerprint extracted from the integrity table.

[0008] Such a solution requires to load an integrity table into the internal memory (e.g. a RAM) of the secure element. For example, in case that the
5 external memory stores 1MB of digital information and the digital information is segmented in pieces of 1KB having each a fingerprint or TAG of 32B, the integrity table to be stored in the internal memory RAM of the secure element should have a storage capacity of 32KB. For a processing device having little storage capacity, like an IoT device, such a solution may not be appropriate
10 because it requires too much storage capacity in the secure element.

[0009] The present disclosure intends to improve the situation.

SUMMARY

[0010] The present disclosure concerns a secure element for securely processing digital information, said digital information being segmented and
15 stored in a plurality of M segments in a memory external to the secure element, including:

- a secure processor for securely processing the plurality of M segments of digital information ;
- a system configured to load a segment of digital information from the
20 external memory into the secure element, and pre-process said segment of digital information by executing a cryptographic algorithm before processing said segment of digital information by the secure processor;

wherein the system is configured to determine an identification information of the segment of digital information that has been loaded, the determination of
25 said identification information including obtaining a version number of said segment of digital information, and to execute the cryptographic algorithm by using said identification information;

characterized in that the system is configured to load a version table containing the respective version numbers of the plurality of M segments of
30 digital information, from the external memory into an internal memory of the

secure element, and to read the version number of each segment of digital information that is loaded in the version table stored in the internal memory.

[0011] Thus, the cryptographic algorithm uses the version number of the digital information stored in the external memory (or a piece of the digital information stored in the external memory) as an input. The version number of the digital information is cryptographically used to pre-process said the digital information (or piece of digital information). It is associated with the digital information (or with the piece of digital information) in the external memory and loaded into the secure element. The version number can have an initial value (e.g. zero) and be incremented each time the digital information (or the piece of digital information) is modified, for example during processing by the secure element. The digital information (or the piece of digital information) is cryptographically bound to its version number. The secure element stores internally this version number, which requires low memory capacity. Thanks to that, the secure element is protected against malicious replay attacks without requiring high capacity storage. So, the present disclosure can be applied to low-end devices that need a good security level.

[0012] Advantageously, the system is configured to determine the identification information of the segment of digital information, the determination of said identification information further including obtaining a segment identifier of said segment of digital information, and to execute the cryptographic algorithm by using said identification information, said segment identifier being either an index assigned to the corresponding segment of digital information in the external memory or an identifier present in a header of the corresponding segment of digital information.

[0013] The identifier of the digital information (or piece of digital information) can be cryptographically used, in addition to its version number, to pre-process the digital information.

[0014] The identifier is a short piece of information, for example an index of the piece of digital information in the external memory. Typically, the digital information is divided in pieces of digital information, that are indexed and stored in the external memory. The pieces of digital information can be
5 stacked or more generally arranged in given positions in the external memory. The indexes that are assigned to the pieces of digital information can depend on the positions of these pieces of digital information in the external memory.

[0015] Advantageously, said secure element stores at least one unique
10 cryptographic key, that was generated uniquely for the secure element, to be used by the system in executing the cryptographic algorithm.

[0016] The use of the version number prevents from a substitution of the digital information (or piece of digital information) by an earlier version of the digital information. The use of the identifier of the digital information (or piece
15 of digital information) prevents from a substitution of the digital information by another valid digital information of loaded in the secure element. The use of a unique key prevents from a substitution of the digital information by a digital information from another secure element or device. The combined use of the version number, the identifier and the unique key allows to have a solution
20 equivalent to the TAG solution of the prior art, which implicitly protect from many different attacks, but without requiring a high storage capacity.

[0017] The system can comprises a key derivation function to generate at least one cryptographic key, to be used by the cryptographic algorithm, by using the at least one unique cryptographic key and the identification
25 information of the digital information as inputs.

[0018] The cryptographic algorithm can comprise an authentication algorithm to authenticate the segment of digital information that has been loaded. Alternatively, or additionally, the cryptographic algorithm can comprise a decryption algorithm to decrypt the digital information that has been loaded.
30 Thus, the version number and/or the identifier of the digital information (or

piece of digital information) can be used not only for the authentication but also for the decryption of the digital information, which increases the security.

[0019] For example, the system is configured to generate an initialization vector, to be used by the decryption algorithm, by using the identification
5 information of the segment of digital information as an input.

[0020] The digital information stored in the external memory can be protected by an authenticated encryption mechanism, for example based on an approach "Encrypt-then-MAC". In that case, it is required to authenticate and then decrypt the digital information, before processing securely the digital
10 information in the secure element. With such a configuration, the use of the version number and the identifier of the digital information as inputs for its authentication provides the highest level of security. Indeed, the digital information will not be decrypted if this authentication is unsuccessful.

[0021] Advantageously, the system is further configured to verify a freshness
15 information of said version table by comparing said freshness information of the version table with a freshness counter stored in an internal non-volatile memory of the secure element.

[0022] The freshness or coherence of all the pieces of digital information is authenticated by the version table. Using the version numbers of the pieces
20 of digital information, instead of their TAGs (fingerprints), allows to spare internal storage resources in the secure element. For example, in case of an external memory storing 1MB of digital information stored in pieces or fragments of 1KB having each a TAG like a MAC of 32B, the size of the integrity table containing the TAGs of all the pieces of digital information
25 would be 32KB. In the present disclosure, if we assume that the version number has a size of a few bytes, for example 3B, the size of the version table that needs to be stored in the secure element is only 3KB. So, the present disclosure allows to spare 29KB of internal storage capacity.

[0023] The verification of the freshness information of the version table
30 ensures the security by building a chain of trust for freshness (anti-replay).

[0024] The internal non-volatile memory can be OTP (One Time Programmable) memory.

[0025] The system can be configured to load the version table in a startup procedure of the secure element.

- 5 [0026] Advantageously, the system is configured to authenticate and decrypt the version table that has been loaded.

[0027] The system can be configured to read the freshness information from a header of the version table.

- 10 [0028] Advantageously, the secure element comprises an update module configured, in case that a segment of digital information (or piece of digital information) is modified when it is processed by the secure processor, to control an operation of writing the modified segment of digital information and an updated version table in the external memory, and incrementing the freshness counter in the internal non-volatile memory.

- 15 [0029] A second aspect of the disclosure concerns a system including the secure element as previously defined and an external memory for storing the digital information.

- [0030] A third aspect of the present disclosure concerns a method for securely processing digital information by a secure element, said digital
20 information being segmented and stored in a plurality of M segments in a memory external to the secure element, including the following steps, performed by the secure element, of:

- loading a segment of digital information from the external memory into the secure element;
- 25 - determining an identification information of the segment of digital information that has been loaded by obtaining a version number of said segment of digital information;
- pre-processing said segment of digital information by executing a cryptographic algorithm using said identification information before

processing said segment of digital information by a secure processor of the secure element;

- securely processing the segment of digital information by the secure processor ;

- 5 characterized in that the method further comprises
a step of loading a version table containing respective version numbers of the plurality of M segments of digital information from the external memory into an internal memory of the secure element and a step of reading the version number of each segment of digital information that is loaded in the version
10 table stored in the internal memory.

[0031] The method can further comprise a step of verifying a freshness information of said version table by comparing said freshness information of the version table with a freshness counter stored in an internal non-volatile memory of the secure element.

BRIEF DESCRIPTION OF THE DRAWINGS

- 15 [0032] Other features, purposes and advantages of the disclosure will become more explicit by means of reading the detailed statement of the non-restrictive embodiments made with reference to the accompanying drawings.

[0033] Figure 1 shows a secure element embedded in a SoC (System on the Chip), according to a first embodiment.

- 20 [0034] Figure 2 shows the secure element and an external memory operating during a startup procedure, in a simplified manner, according to the first embodiment.

- [0035] Figure 3 shows the secure element and the external memory operating during a procedure of loading and pre-processing (or reading) the digital
25 information from the external memory, in a simplified manner, according to the first embodiment.

[0036] Figure 4 shows the secure element and the external memory operating during a procedure of writing a modified piece of digital information and an updated version table into the external memory, in a simplified manner, according to the first embodiment.

- 5 [0037] Figure 5 shows a flowchart of a method of securely processing digital information, performed by the secure element, the digital information being stored in the external memory, according to the first embodiment.

[0038] Figure 6 shows a flowchart of an update procedure executed when a piece of digital information has been modified by the secure element,
10 according to the first embodiment.

[0039] Figure 7 represents schematically the secure element according to the first embodiment.

DETAILED DESCRIPTION

[0040] Figure 1 shows a system including a secure element 100 and an external memory 200 (i.e., a memory external to the secure element 100),
15 according to a first embodiment. The secure element 100 is for example a smart card or a chip. It can be embedded in a SoC (System on Chip) 300 having several processing units, several memories and several functionalities (not represented). The SoC 300 may be integrated in a larger module 400. The external memory 200 is external to the secure element 100. It can be
20 built on a different silicon substrate of a secure processor of the secure element 100. For example, the external memory is arranged in the module 400, outside the SoC 300. However, the external memory 200 could be disposed in the SoC 300. For example, this larger module 400 could be an IoT device (typically provided with a unique identifier and the ability to transfer
25 data over a network), a telecommunication apparatus, a location system, a vehicle like a car or a plane, etc. Examples of IoT devices include smart meters, smart cameras, sensors, trackers, tags, detectors, monitors, wearable items and clothing, smart home devices, medical and health-care

devices, life science devices, set-top boxes, and edge devices in telecommunication networks such as 5G networks.

[0041] Different illustrative use cases (not limitative) of the secure element 100 are given below.

- 5 [0042] In a first use case, the secure element 100 can be integrated in a modem of a telecommunication system or apparatus. In such a case, the secure element 100 can handle network authentication and download secure applications.

- 10 [0043] In a second use case, the secure element 100 can be integrated in a tachograph and securely handle location data.

[0044] In a third use case, the secure element 100 can be integrated in a vehicle, for example a car or a plane, to secure and manage safety data transport.

- 15 [0045] The external memory 200 stores digital information. The terms "digital information" designate data liable to be loaded into the secure element 100, such as executable code or information generated by executable code or used by executable code, or any other data to be used or processed by the secure element 100.

- 20 [0046] The secure element 100 is intended to load, pre-process (or read) and securely process the digital information stored in the external memory 200, as explained later in the description.

[0047] The external memory 200 can be a non-volatile memory.

- 25 [0048] In the present embodiment, memory segmentation is used in the external memory 200. It means that the digital information is segmented and stored in segments (also called 'fragments'). A segment of digital information is a piece of digital information resulting from a segmentation. The segmentation is used to store, transfer and pre-process the digital information.

[0049] For example, the external memory 200 stores M segments of digital information referred as ' S_i ', with $1 \leq i \leq M$. The M segments S_i can include M_1 segments of code and M_2 segments of data, with $M_1 \geq 0$ and $M_2 \geq 0$. The segments of digital information can have the same size or have
5 respective sizes that may be different, depending on the implementation. In some embodiments, the secure element 100 has a cache implementation. In that case, the secure element 100 can have a cache controller (typically a cache hardware component) configured to load and store cache lines (corresponding to segments of digital information) of a predetermined size. In
10 other embodiments, the segments of digital information could be pieces of digital information requested by the secure processor, when executing a software component or an application. In such a case, the segments could have different respective sizes.

[0050] The segments of digital information S_i are protected by an
15 authenticated encryption algorithm to ensure security during storage in the external memory 200 and during transfer from the external memory 200 to the secure element 100. In the first embodiment, the authenticated encryption used to protect the digital information is based on the well-known "Encrypt-then-MAC" (EtM) approach. The EtM approach is considered as a very
20 robust approach for authenticated encryption. For each segment of digital information S_i , the external memory 200 stores the element $[S_i] \parallel \text{MAC}_{[S_i]}$, containing the segment S_i encrypted with an encryption key (described later) and concatenated with the authentication element $\text{MAC}_{[S_i]}$ of the encrypted segment $[S_i]$ computed with an authentication key (also described later). The
25 authentication element $\text{MAC}_{[S_i]}$ can be calculated by a MAC function.

[0051] In the present disclosure, the brackets ' $[\]$ ' represent the encrypted form of an element and the symbol ' $\parallel$ ' represents the concatenation of two elements.

[0052] A version number V_i is attributed to each segment of digital information
30 S_i and incremented when this segment of digital information S_i is modified by

the secure element 100. The size of the version number V_i of one segment of digital information S_i can be small, generally a few bytes, for example 3 bytes. Initially, when the digital information is stored in the external memory 200, the version numbers V_i of all the segments S_i of digital information with
5 $1 \leq i \leq M$ can be set to an initial value, for example zero (but it could be one or any other value). Then, each time one specific segment (here referred as 'Sj') among the M segments S_i is modified, its version number V_j is incremented, for example by one (i.e., $V_j = V_j + 1$).

[0053] In addition, in the first embodiment, each segment of digital information
10 has an associated segment identifier referred as ID_{S_i} . This segment identifier ID_{S_i} is a short piece of information that identifies the segment. The segment identifier ID_{S_i} can be an index of the segment S_i in the external memory 200. Typically, indexes are assigned to the segments of digital information S_i in the external memory 200, for example depending on the respective positions
15 of the segments S_i stored in the external memory 200. These indexes can be used as segment identifiers ID_{S_i} by the secure element 100. They can also be used to index the version numbers V_i of the segments S_i in the version table. In case of a cache implementation in the secure element 100, cache block addresses are given by addresses to fetch code or load data and can be
20 used as segment (or block) identifiers ID_{S_i} . These cache block addresses correspond to the indexes of the blocks (segments) in the external memory 200. The same cache block addresses can be used to index the version numbers in the version table.

[0054] In the present embodiment, the version number V_i and the identifier
25 ID_{S_i} of each segment of digital information S_i stored in the external memory 200 are used in the authenticated encryption algorithm as inputs. For example, the authentication algorithm, such as a MAC algorithm, used to authenticate the segment S_i , uses an authentication key $k1_{S_i}$ that is derived from a master key $k1$ (or source key $k1$) by a key derivation function (KDF)
30 that takes the version number V_i and the identifier ID_{S_i} of the segment S_i as

inputs, as expressed below :

$$k1_Si = KDF(k1, Vi, ID_{Si})$$

[0055] Additionally, in order to increase the security, the version number V_i and the identifier ID_{Si} of the segment S_i can also be used to encrypt the segment of digital information S_i . For example, the encryption algorithm uses an encryption key $k2_Si$ that is derived from a master key $k2$ (or source key $k2$) by a key derivation function (KDF) that takes the version number V_i and the identifier ID_{Si} of the segment S_i as inputs:

$$k2_Si = KDF(k2, Vi, ID_{Si})$$

[0056] The keys $k1$ and $k2$ can be secret keys. Advantageously, these keys $k1$ and $k2$ are generated uniquely for the secure element 100. They can be generated by the secure element 100 itself. The keys $k1$ and $k2$ are different in the first embodiment. However, they could be the same key in other embodiments.

[0057] All the encrypted and authenticated segments $[S_i] \parallel MAC_{[S_i]}$ (with $1 \leq i \leq M$) are stored in the external memory 200. They can be stacked (and ordered) in a memory area of the external memory 200.

[0058] The external memory 200 further stores a version table, referred as "CtrlObj" in figures 2-4. The version table CtrlObj contains the respective current (valid) version numbers V_i of all segments of digital information S_i (with $1 \leq i \leq M$) stored in the external memory 200. The version numbers V_i contained in the version table CtrlObj can be arranged in the same order as the segments S_i are arranged in the external memory 200. More generally, the version numbers V_i in the version table CtrlObj can be indexed in the same manner as the corresponding segments S_i are indexed in the external memory 200. This allows to identify the segments S_i by their ordering numbers or indexes. Alternatively, the segments of digital information could have segment identifiers in a segment. These segment identifiers present in the headers of the segments could be associated with the version numbers V_i in the version table CtrlObj.

[0059] Furthermore, a freshness information is attributed to the version table CtrlObj. Typically, this freshness information is a version number of the version table, referred as 'CtrlObj_Version'. This freshness information has an initial value, for example zero, that is attributed to the version table CtrlObj
5 when the digital information is initially stored in segments in the external memory 200. It can be incremented each time the content of the version table CtrlObj is modified, as will be explained later. The freshness information CtrlObj_Version of the version table CtrlObj is stored in the external memory 200, for example in a header of the version table CtrlObj.

10 [0060] The version table CtrlObj can also be protected for its storage in the external memory 200 and its transfer between the secure element 100 and the external memory 200, for example by using an authenticated encryption algorithm. For example, the version table CtrlObj is encrypted with the key k1 and then an authentication element of the version table CtrlObj (here in
15 encrypted form) is computed with a authentication algorithm, such as a MAC algorithm, using the key k2. The external memory 200 stores the encrypted version table $[\text{CtrlObj}]_{k1}$ and its authentication element $\text{MAC}_{[\text{CtrlObj}]}$. Any other authenticated encryption algorithm may be used.

[0061] The secure element 100 comprises a secure central processing unit
20 CPU (or secure processor) 110, a plurality of internal memories 121, 122, 130 and a loading and pre-processing system 140.

[0062] The secure CPU 110, also known as a secure processor, has security features having a high assurance level certification for security-critical applications. It can be an integrated hardware intellectual property core (IP
25 core).

[0063] A first internal memory 121 is for storing segments of digital information S_i before processing by the central processing unit 110. It is for example a cache memory.

[0064] A second internal memory 122 is for storing the version table CtrlObj, of the segments of digital information requested by the secure CPU 110. It can be a volatile memory, such as a RAM.

5 [0065] The first and second internal memories 121, 122 are part of an internal volatile memory system 120. They can be separate memories or different areas of one memory.

[0066] A third internal memory 130 is a persistent (non-volatile) memory for storing data like keys and counters. More precisely, the internal memory 130 can store keys, like the master keys k1 and k2, to be used in the
10 cryptographic algorithms executed by the loading and pre-processing system 140 or by the secure processor 110 and the freshness counter 'FC' related to the version table CtrlObj. The persistent internal memory 130 can be an OTP (One Time Programmable) non-volatile memory.

[0067] The loading and pre-processing system 140 has several functions
15 including:

- the function of loading the version table CtrlObj from the external memory 200 into the secure element 100,
- the function of loading pieces (segments) of digital information from the external memory 200 into the secure element 100,
- 20 - the function of pre-processing the version table CtrlObj to be stored in the second internal memory 122 (RAM),
- the function of pre-processing the pieces (segments) of digital information to be stored in the first internal memory 121 (cache memory) and to be processed by the secure processor 110, and
- 25 - the function of checking the freshness of the loaded version table CtrlObj.

[0068] These functions of the loading and pre-processing system 140 are implemented by components 141-145, described below. They can be hardware components, or software components, or by a mix of hardware components and software components. The secure processor 110 is
30 arranged to execute the software components. In figure 7, the components

141-145 are represented outside the secure processor 110 but they could be arranged at least partially inside the secure processor 110 (in particular the software components).

[0069] The loading and pre-processing system 140 has a first loader 141, a
5 second loader 142, an authentication verification module 143, a decryption module 144 and a freshness checking module 145 (or anti-replay checking module).

[0070] The first loader 141 is for loading the version table CtrlObj from the external memory 200. The second loader 142 is for loading segments of
10 digital information Si from the external memory 200.

[0071] The authentication verification module 143 is configured to authenticate loaded data, such as the version table CtrlObj and segments of digital information Si. In the present embodiment, it implements a MAC function and a key derivation function KDF to derive keys from the master key
15 k1, using the version numbers and the identifiers of these segments as inputs, to be used as authentication keys to authenticate the segments of digital information.

[0072] The decryption module 144 is configured to decrypt loaded data, such as the version table CtrlObj and segments of digital information Si. In the
20 present embodiment, it implements a decryption algorithm and a key derivation function KDF to derive keys from the master key k2, using the version numbers and the identifiers of these segments as inputs, to be used as decryption keys to decrypt the segments of digital information.

[0073] Alternatively, the key derivation function could be implemented in one
25 separate component 146 (represented by a dashed block in figure 7), used by both the authentication verification module 143 and by the decryption module 144.

[0074] The freshness checking module 145 is configured to check the freshness of the version table CtrlObj with respect to the internal freshness

counter FC in the OTP memory 130, when this version table CtrlObj is loaded from the external memory 200, as will be described later in the description of the method of securely processing digital information.

[0075] The secure element 100 can further include an update module 150 for
5 updating the external memory 200, when the secure processor 110 has modified one or more segments of digital information, as explained later in the description.

[0076] A host processor or CPU (not represented) could be provided outside the secure element 100, for example in the SoC 300, to execute some
10 operations under control of the secure element 100, for example loading the version table or the segments. Thus, loading could be performed through a system bus or a host CPU, upon request of the secure element 100.

[0077] The secure element 100 has other cryptographic functions, including a function of cryptographic encryption and an authentication function like a
15 MAC function, that allow the secure CPU 110 to encrypt and authenticate data (typically segments of digital information and the version table CtrlObj), as explained later. The function of cryptographic encryption and the authentication function are implemented by components 148, 149. The latter are either software components, or hardware components, or a mix of
20 hardware and software components.

[0078] The secure processor, or secure central processing unit (CPU), 110 is configured to securely process or use the pieces of digital information loaded in the internal (cache) memory 121. It is intended to execute security-critical applications. The secure processor 110 is also configured to control the
25 operations of the secure element, such as the authenticated decryption and the authenticated encryption. In addition, when a segment of digital information has been modified during processing, the secure CPU 110 can control the update of the version table CtrlObj in the internal memory 122 and also the update of the persistent memories 200 and 130 (i.e., the update of
30 the modified segment(s) and the version table CtrlObj in the external memory

200 and the update of the freshness counter FC in the internal memory 130) that has to be carried out by an atomic transaction.

[0079] The update module 150 has the function of incrementing the freshness counter in the OTP memory 130 and updating the external memory 200, when a segment of digital information has been modified during processing, under control of the secure processor 110, as will be explained later. It can be based mostly on software. The action of incrementing the freshness counter in the internal OTP memory 130 and the action of updating the external memory 200 by writing the modified segment of digital information and the updated version table in the external memory 200 are advantageously executed together by a single atomic transaction, for functional reasons.

[0080] The secure element 100 has also a hardware interface 160 to directly access the external non-volatile memory 200.

[0081] The components of the secure element 100 are advantageously trusted, which means that they cannot be manipulated to induce faults.

[0082] The method of securely processing digital information stored in the external memory 200 by the secure element 100 will now be described with reference to the figures 2-6. The method can be divided into three procedures:

- a startup procedure, that is executed after switching on the secure element 100 (figures 2 and 5);
- a reading or pre-processing procedure, that follows the startup procedure and is executed before securely processing digital information by the secure processor 110 (figures 3 and 5); and
- a updating procedure, that is executed when a piece of digital information is modified during processing by the secure processor (figures 4 and 6).

Startup procedure (figures 2 and 5)

[0083] Initially, the secure element 100 is switched on. Subsequently, a booting or startup procedure is executed by the secure element 100. During

this startup procedure, the secure element 100 can execute a security protocol (e.g., secure boot) that can secure the boot process. In the startup procedure, the first loader 141 loads the version table CtrlObj read from the external memory 200, in a step S50. More precisely, the version table CtrlObj
5 in encrypted form and concatenated with its MAC, $[\text{CtrlObj}]_{k1}|\text{MAC}_{[\text{CtrlObj}]}$, is loaded from the external memory 200 to the secure element 100. Then, the authentication verification module 143 authenticates the version table CtrlObj, in a step S51. To this end, the authentication verification module 143 compute a MAC of the encrypted version table loaded, $[\text{CtrlObj}]_{k1}$, using the
10 key k2, and compares the computed MAC with the MAC loaded with the version table $\text{MAC}_{[\text{CtrlObj}]}$. If the authentication is successfully achieved, the encrypted version table $[\text{CtrlObj}]_{k1}$ is decrypted by the decryption module 144, in a step S52, using the key k1. If the authentication is unsuccessful, the step S51 is followed by an error step S70. For example, in the error step S70, the
15 method for processing the digital information is aborted (interrupted).

[0084] After authentication with the key k2 and decryption with the key k1 of the version table CtrlObj, the freshness of the version table CtrlObj is checked by the freshness verification module 145, in a step S53. To this end, the freshness information CtrlObj_Version, included in the header of the
20 version table CtrlObj, is compared to the freshness counter FC stored in the OTP memory 130.

[0085] If the freshness information CtrlObj_Version of the version table CtrlObj matches the freshness counter FC stored in the OTP memory 130, the freshness of the version table CtrlObj is successfully checked and the
25 method proceeds with the steps S54 to S59 of loading and pre-processing segments of digital information Si. If the check of the freshness of the loaded version table is unsuccessful, the method goes to an error step S71. For example, in the error step S71, the method for processing the digital information is aborted (interrupted).

30 Reading or pre-processing procedure (figures 3 and 5)

[0086] The startup procedure is followed by a reading or pre-processing procedure, performed by the secure element 100, for reading or pre-processing segments of digital information from the external memory 200.

[0087] The loading and pre-processing procedure includes the steps S54 to S59 (described below) that are repeated iteratively for each of a plurality of segments of digital information S_i , with $1 \leq i$. During the loading and pre-processing procedure, the index 'i' represents the order in which some segments of digital information are successively fetched in the external memory 200 and then pre-processed by the secure element 100 over time.

10 This order of fetching (or the order of loading) does not necessarily correspond to the order in which the segments of digital information are stacked in the external memory 200. In other words, when S_i is assumed to be a segment of digital information currently loaded and pre-processed by the secure element 100, S_{i+1} is the next segment of digital information to be loaded and pre-processed. But S_i and S_{i+1} may not be consecutively stacked segments in the external memory.

[0088] In the step S54, a segment of digital information S_i in protected form is loaded. More precisely, the second loader 142 loads the element $[S_i]||MAC_{[S_i]}$ containing the segment of digital information S_i encrypted with the key $k1_S_i$ and concatenated with the authentication element $MAC_{[S_i]}$ of the encrypted segment $[S_i]$ computed with the key $k2_S_i$.

20

[0089] In the step S55, the authentication verification module 143 obtains the version number V_i from the version table $CtrlObj$ stored in the internal memory 122. It also obtains the segment identifier ID_{S_i} from the version table $CtrlObj$ as this ID_{S_i} corresponds to the index of the version number V_i in the version table $CtrlObj$. Then, in a step S56, the authentication verification module 143 authenticates the encrypted segment $[S_i]$, by computing a MAC of $[S_i]$ and comparing it with the $MAC_{[S_i]}$ loaded with $[S_i]$. To compute this MAC, the module 143 derives the key $k2_S_i$ from the master key $k2$ read in the memory 130 and uses the version number V_i and the identifier ID_{S_i} of the

25

30

segment S_i (read in the step S55) as inputs of the key derivation function. If the authentication is successfully achieved (i.e., the computed MAC matches the loaded MAC), the encrypted segment $[S_i]$ is then decrypted by the decryption module 143, in a step S58. The decryption uses a key $k1_Si$
5 computed from the master key $k1$ (read from the memory 130), and uses the version number V_i and the identifier ID_{Si} of the segment S_i , obtained from the version table $CtrlObj$ stored in the memory 122 in the step S57, as an input of the key derivation function (KDF). If the authentication is unsuccessful, the step S56 is followed by an error step S72. For example, in the error step S72,
10 the method for processing the digital information is aborted (interrupted). Instead, the segment of digital information S_i could be deleted.

[0090] During the pre-processing procedure, in the steps S55 and S57, the loading and pre-processing system 140 obtains the version number V_i and the identifier ID_{Si} of the segment of digital information S_i from the internal
15 memory 122 (e.g. RAM), and provides this identification information of the segment S_i as an input in executing the cryptographic algorithms of authentication and decryption used to pre-process the segment S_i . Alternatively, the identification information of the segment S_i from the version table $CtrlObj$ stored in the internal memory 122 may be used by only one of
20 the two cryptographic algorithms, preferably by the authentication algorithm. However, the security is increased if decryption uses the identification information of the segment S_i as well.

[0091] Instead of using the segment version number and the segment identifier as inputs to generate the decryption key, the version number V_i and
25 the segment identifier ID_{Si} may be used to generate an initialization vector IV to be used by the decryption module 144 for decrypting the segment of digital information $[S_i]$.

[0092] After authentication and decryption of the segment of digital information S_i , the segment S_i is loaded into the cache memory 121 of the
30 secure processor 110 in a step S59.

[0093] Then, in a step S60, the segment of digital information S_i is accessible by the secure processor 110 and can be securely processed by the secure processor 110. For example, the secure processor executes a secure (or security-critical) application by processing the segment of digital information S_i .

[0094] The steps S54 to S59 are repeated iteratively for segments of digital information S_i successively loaded from the external memory 200 into the secure element 100, unless the process is aborted beforehand due to an error.

10 Update procedure (figures 4 and 6)

[0095] When a segment of digital information (here referred as ' S_j ' with $1 \leq j \leq M$) is processed by the secure processor 110, it might be modified during its processing, in a step S80. The modified version of the segment of digital information S_j is referred as S_j' . In such a case, an update procedure is achieved and will now be described with reference to figures 4 and 6.

[0096] When the initial segment S_j is changed into the modified segment S_j' , the secure processor 110 updates the version table CtrlObj stored in the internal memory 122 by incrementing the version number of the modified segment S_j' in the version table CtrlObj stored in the internal memory 122, in a step S81. If we assume that the version number of the initial segment S_j is V_j , the new version number of the modified segment S_j' is V_j+1 . In the step S81 of updating the version table CtrlObj in the internal memory 122, the secure processor 110 also updates the version number of the version table CtrlObj, CtrlObj_Version, here in the header of the version table CtrlObj stored in the internal memory 122. The version number of the version table CtrlObj, CtrlObj_Version, is incremented by one (CtrlObj_Version=CtrlObj_Version+1). The updated version table (stored in the internal memory 122) is referred as CtrlObj'.

[0097] The secure element 100 also needs to update the external memory 200. More precisely, it needs to write the modified piece of digital information

S_j' and the updated version table $CtrlObj'$ in the external memory 200. To this end, the secure processor 110 submits the updated version table $CtrlObj'$ and the modified segment S_j' to an authenticated encryption algorithm, in a step S82, before controlling a transfer of the modified segment S_j' and updated version table $CtrlObj'$ to the external memory 200. The same authenticated encryption algorithm as the one used to prepare the data stored in the external memory is used. In the present embodiment, the authenticated encryption algorithm follows an Encrypt-then-Mac approach. The encryption of the version table $CtrlObj'$ uses the key k_1 , and the MAC function applied to $[CtrlObj']_{k_1}$ uses the key k_2 . The encryption of the modified segment of digital information S_j' uses the derived key $k1_Vj+1$, where $k1_Vj+1 = KDF(k_1, Vj+1, ID_{S_j})$. It should be noted that the segment identifier of the modified segment S_j' is the same as the segment identifier of the corresponding segment S_j (before modification). The authentication of the modified segment of digital information in encrypted form $[S_j']$ uses the derived key $k2_Vj+1$, where $k2_Vj+1 = KDF(k_2, Vj+1, ID_{S_j})$.

[0098] Then, in a step S83, under control of the secure processor 110, the update module 150 increments the freshness counter FC of the version table in the OTP memory 130, and writes the new version table $CtrlObj'$ and the amended segment of digital information S_j' in the external memory 200. The freshness counter FC is here incremented by one. All these actions (incrementing FC in the internal memory 130 and writing $CtrlObj'$ and S_j' in the external memory 200) are executed together by a single atomic transaction. Such an operation has only two persistent states: an original state without the modification and a final state with the modification. The operation can be quick or take a long time.

[0099] A procedure analogous to the above update procedure can be carried out initially by the secure element 100 to store the segments of digital information in encrypted and authenticated form in the external memory 200. The secure element 100 can have a firmware secure loader configured to

securely load the digital information to be stored in the external memory from a digital information provider, for example through a communication network or through a local connection to another machine. The secure element can be configured to

- 5 - segment the loaded digital information in segments of digital information, authenticate and encrypt these segments (in the same manner as described in the update procedure for the modified segment S_j) and write them in the external memory,
- generate a version table $CtrlObj$, authenticate and encrypt this version table
10 (in the same manner as described in the update procedure) and then write it into the external memory.

[0100] A second embodiment is based on the first embodiment and only differs from it by the authenticated encryption algorithm. In the second exemplary embodiment, the authenticated encryption follows an approach of the type
15 "MAC-then-Encrypt-then-MAC". This approach consists, for each segment of digital information S_i , in computing a first MAC ' MAC_{i1} ' of the segment S_i in clear using a first key $k0_Vi$ (derived from a master key $k0$ and using the version number Vi as an input), then encrypting the segment S_i and the first MAC concatenated together (i.e., $S_i || MAC_{i1}$) with a second key $k1_Si$ (derived
20 from a master key $k1$ and using the version number Vi as an input), and then computing a second MAC ' MAC_{i2} ' of the result of the encryption (i.e., $[S_i][MAC_{i1}]$) using a third key $k2_Si$ (derived from a master key $k2$ and using the version number Vi as an input), to obtain the protected segment $[S_i][MAC_{i1}]MAC_{i2}$.

25 [0101] A third embodiment is based on the first or the second embodiment and differs from them only in that the identification information of the segment of digital information used in the authentication operation and/or in the decryption operation as an input, only contains the version number of the segment of digital information. Such an embodiment is appropriate for

example in case that the secure element loads only one segment or piece of digital information.

- [0102] Different types of MAC function could be used. For example, a CMAC function (Cipher-based Message Authentication Code), that is a block cipher-based message authentication code algorithm, or a HMAC algorithm (hash-based message authentication code), that is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key, could be used. Instead of a MAC function, any other authentication function could be used.
- 5