(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 390 796**

(51) Int. CI.:
G06F 21/22 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Número de solicitud europea: **09166439 .1**
(96) Fecha de presentación: **27.07.2009**
(97) Número de publicación de la solicitud: **2280365**
(97) Fecha de publicación de la solicitud: **02.02.2011**

(54) Título: **Método de implementación de un procesador para garantizar la integridad de un software**

(45) Fecha de publicación de la mención BOPI:
16.11.2012

(45) Fecha de la publicación del folleto de la patente:
16.11.2012

(73) Titular/es:
NAGRAVISION S.A. (100.0%)
Route de Genève 22-24
1033 Cheseaux-sur-Lausanne, CH

(72) Inventor/es:
MACCHETTI, MARCO y
KUDELSKI, HENRI

(74) Agente/Representante:
TOMAS GIL, Tesifonte Enrique

DESCRIPCIÓN

Método de implementación de un procesador para garantizar la integridad de un software

INTRODUCCIÓN

[0001] La presente invención se refiere al campo de protección de software y más particularmente a un dispositivo y a un medio de protección de un software, destinados a garantizar la integridad de una unidad de software.

ESTADO DE LA TÉCNICA

[0002] En el campo del tratamiento seguro de datos, se debe proveer un entorno inviolable al interior del cual el tratamiento se puede desarrollar de forma segura. Un primer enfoque para luchar contra el problema de seguridad de aplicaciones se centraba en el esfuerzo por hacer que el hardware en el que se alojaba el software fuera lo más seguro posible. La noción de protección contra manipulaciones significaba entonces que dicho hardware era difícil de abrir o que una vez abierto, se destruiría el chip sobre el cual residía el software seguro. No obstante, se ha comprobado actualmente que de forma general las técnicas de software que permiten obtener una seguridad de aplicación ofrecen más flexibilidad y costes inferiores y de hecho en la mayoría de los casos en los que una buena seguridad de aplicación implica la garantía de que una unidad de software no ha sido manipulada, se utiliza una combinación de enfoques de software y de hardware.

[0003] Un sistema típico sobre el que funciona una aplicación comprende generalmente una unidad de procesamiento, una pluralidad de periféricos y una memoria. En la mayoría de los casos en los que se requiere seguridad, se utilizan esquemas de cifrado. En tales esquemas, la información que se debe proteger, es decir los datos a tratar o un código ejecutable, es cifrada. El cifrado se realiza normalmente dentro de un módulo de seguridad que forma parte del sistema. El módulo de seguridad se puede instalar de varias maneras, por ejemplo en una tarjeta de microprocesador, una tarjeta inteligente o en cualquier módulo electrónico en forma de etiqueta de identificación o de clave. Estos módulos son generalmente portátiles y amovibles del receptor y se diseñan para ser inviolables. La forma más comúnmente usada tiene contactos eléctricos, pero también existen versiones sin contacto de tipo ISO 14443.

[0004] Otra implementación del módulo de seguridad existe en la que este último es soldado directamente dentro del receptor, una variación de esto siendo un circuito en un enchufe o un conector tal como un módulo SIM. Otra implementación más consiste en integrar el módulo de seguridad en un chip que tenga otra función, por ejemplo en un módulo de desaleatorización o en un módulo de microprocesador de un decodificador. El módulo de seguridad también se puede instalar en el software.

[0005] A pesar del uso de módulos de seguridad y de técnicas de cifrado avanzadas en los sistemas modernos actuales de tratamiento seguro, tales sistemas siguen siendo vulnerables a los intentos de violación de la seguridad. Las técnicas utilizadas para violar la seguridad de tales sistemas incluyen por ejemplo, la ingeniería inversa del hardware implicado o el análisis estático o dinámico del software usado allí y la manipulación consecuente con dicho software. Se hace referencia con análisis estático a alguna forma de desensamblaje o de descompilación de código no aplicable. Se hace referencia con análisis dinámico a un análisis llevado a cabo durante el uso del código, es decir mediante la observación de ciertas señales durante el uso del software. Tales análisis pueden conducir a la manipulación fraudulenta mediante la cual se modifica el software, por ejemplo, mediante un ataque por interferencia de rama donde un salto incondicional se introduce en lugar de un salto condicional de manera a forzar la ejecución de una rama cuando las condiciones actuales no prescriben dicha ejecución. Típicamente, tal ataque forzaría una programa a evitar una fase de autentificación como un número de serie o un control de contraseña por ejemplo.

[0006] En un documento titulado "Resistencia frente a manipulaciones fraudulentas para la Protección de un Software", presentado en 2005 en una tesis para la obtención de un Master en Ciencias, Ping Wang describe una técnica de cifrado por bloque múltiple, donde una programa de software se divide en varios bloques independientes según el flujo del programa. Cada bloque del programa es cifrado después, y cada bloque posee una clave de cifrado diferente. La clave de cifrado para cada bloque es el valor hash del bloque precedente según el flujo del programa. Esta técnica funciona en programas que tienen una estructura en forma de árbol donde los bloques se disponen de forma jerárquica con un bloque conduciendo a otro. En esta técnica, el primer bloque a ejecutar debe estar en lenguaje claro. Un código para llamar a la rutina de descifrado se dispone al interior de cada uno de los bloques y un controlador de programa para implementar el control de integridad dinámica se añade al final del programa. Si un adversario intenta cambiar una parte del programa, entonces el valor hash para el bloque que contiene la parte cambiada del programa será diferente y así el bloque siguiente no será descifrado correctamente y el programa no funcionará.

[0007] Este esquema presenta en consecuencia la desventaja de que cada bloque debe ser leído dos veces. Otra desventaja es que el cifrado se efectúa en un bloque por base de bloque en vez de una instrucción por base de instrucción, con una clave de descifrado válida para un bloque entero. Esto significa que el descubrimiento de una clave vuelve vulnerable un bloque entero de software. El tamaño más pequeño del bloque se determina por el bloque más pequeño que incluye enteramente un bucle, ya que en este diseño, por definición, un bloque debe contener un bucle completo. Aunque se pudiera reducir un programa a una sola instrucción por bloque en caso de no tener ningún bucle,

la sobrecarga resultante de la implementación del método produciría un resultado final difícil de manipular en términos de tamaño y de velocidad de ejecución. Además, se puede imaginar un ataque eventual donde se aplique una modificación a un bloque y un cambio correspondiente se aplique al controlador de programa para compensar la modificación de modo que éste calcule el valor hash apropiado con respecto a la modificación aplicada al bloque preservando así la integridad percibida del programa.

[0008] El documento EP-0908810-A2 divulga un procesador seguro con una memoria externa usando un encadenamiento de bloques y un reordenamiento de bloques donde la información de autentificación es XORed con el último bloque de datos en claro (por ejemplo la información de programa) y se descifra opcionalmente para producir un valor de verificación, con el fin de permitir la verificación de la integridad de cada bloque.

[0009] La presente invención permite que el código ejecutable exista en formato cifrado, donde el cifrado se efectúa en una instrucción por una base de instrucción y no requiere que las instrucciones sean leídas dos veces. El esquema se puede realizar completamente en el hardware con la ventaja inherente de que las claves de cifrado no aparecen nunca en un sitio donde puedan ser vulnerables y interceptadas. No hay sobrecarga de software y por lo tanto la velocidad de ejecución aumenta de forma importante. En la técnica anterior, la clave de cifrado para el siguiente bloque depende sólo del contenido de un bloque precedente. En la presente invención la clave de cifrado puede depender de una acumulación de varios valores de claves de cifrado precedentes. Por ejemplo, la clave de descifrado de la próxima instrucción se puede basar en la instrucción actual combinada con una acumulación de las claves para las dos instrucciones precedentes.

BREVE RESUMEN DE LA INVENCIÓN

[0010] La presente invención tiene como objetivo resolver el problema de la seguridad causado por el análisis de software y la modificación fraudulenta subsecuente de dicho software, mientras que se minimiza la sobrecarga con el fin de realizar la solución y que ésta se vuelva flexible y se pueda aplicar a sistemas que usan un software de muchos tipos de estructura diferentes. Esto se consigue mediante el uso de un método implementado por un procesador para asegurar la integridad de un software en una memoria de programa, dicho software comprendiendo una pluralidad de instrucciones cifradas, una instrucción comprendiendo al menos un opcode, dicho método usando una clave de instrucción inicializada y comprendiendo las siguientes fases de:

- lectura de una instrucción corriente cifrada,

- uso de la clave de instrucción para descifrar la instrucción corriente cifrada,

- actualización de la clave de instrucción mediante un cálculo basado en el valor corriente de la clave de instrucción y en un resumen de la instrucción corriente, de modo que la próxima instrucción cifrada a leer se puede descifrar con la clave de instrucción actualizada,

- ejecución de la instrucción corriente.

[0011] La invención se puede aplicar a programas cuya estructura no se presenta necesariamente en forma de árbol y se puede realizar en el software o enteramente en el hardware de manera a eliminar la posibilidad de intercepción por parte de terceros de una instrucción descifrada o de una clave de descifrado.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

[0012] La presente invención se entenderá mejor en referencia a la siguiente descripción detallada de unas formas de realización preferidas leídas conjuntamente con los dibujos anexos, en los que:

- la Fig. 1 es un diagrama de bloques simplificado de una forma de realización de la presente invención.

- la Fig. 2 muestra un diagrama de flujo de una forma de realización de la presente invención.

- la Fig. 3 es un diagrama de bloques simplificado que muestra cómo los saltos o ramas de software se pueden manipular según una forma de realización de la presente invención.

DESCRIPCIÓN DETALLADA

[0013] Como se ha mencionado anteriormente, la presente invención tiene como objetivo proveer un medio para el funcionamiento de un software de manera segura de modo que el software se memoriza de forma cifrada y se descifra y ejecuta dentro de un procesador seguro según una instrucción por una base de instrucción, lejos de la posibilidad de ser controlado. La clave para el descifrado de una instrucción corriente depende al menos de una instrucción precedente que se ha descifrado correctamente, mientras que la clave para el descifrado de una instrucción siguiente depende del

descifrado adecuado de la instrucción corriente. Se consigue así un medio de auto-control que permite garantizar la integridad de una unidad de software. La simple ejecución exitosa del software es una garantía de que no se han manipulado de forma fraudulenta ni el flujo ni el contenido, ya que una modificación hecha a una instrucción anularía la capacidad de descifrar la instrucción siguiente, lo que llevaría al final prematuro del programa o al menos a una corrupción de la traza de ejecución del programa. El esquema usado en la presente invención se puede realizar en el software, pero se debe tener en cuenta de que éste se puede realizar completamente en el hardware, eliminando así la posibilidad de intercepción por parte de terceros de instrucciones en claro o de intercepción de cualquiera de las claves de descifrado implicadas. La invención casi no produce ninguna sobrecarga en comparación con soluciones del estado del arte. El esquema se puede aplicar a un software de varias arquitecturas o estructuras diferentes incluyendo aquellas con saltos y pausas y no se limita a estructuras conocidas como estructuras arborescentes.

[0014] La presente invención provee en consecuencia un método que permite asegurar la ejecución de un software inviolable en un sistema comprendiendo al menos una memoria de programa (PMEM) para mantener instrucciones de programa cifradas (INSTP', INSTC', INSTF'), un módulo de descifrado (DECR) para descifrar dichas instrucciones de programa, una unidad de tratamiento de datos (SCPU) para ejecutar las instrucciones de programa descifradas (INSTP, INSTC, INSTF) y un medio de construcción de claves de descifrado, conocidas como claves de instrucción (KP, KC, KF), para descifrar las instrucciones de programa cifradas. Los medios de construcción de las claves de instrucción por supuesto pueden residir en la unidad de tratamiento de datos. El módulo de descifrado y la unidad de tratamiento de datos residen preferiblemente dentro de un módulo de seguridad de cualquier tipo conocido en el estado de la técnica.

[0015] Durante la ejecución del programa cifrado, la instrucción corriente cifrada (INSTC') es leída a partir de la memoria de programa (PMEM) y es descifrada (DECR) para producir una instrucción corriente (INSTC) mediante el uso de una clave de descifrado corriente (KC) que se construye a partir de una combinación (Fn), por un lado de un resumen de la clave de descifrado precedente (KP) y por otro lado de un resumen de la instrucción ejecutada previamente (DIG(INSTRP)), como mostrado en la Fig. 1. Con "resumen" se hace referencia a cualquier operación aplicada a todo o a una parte de un operando y a la producción de una salida. Es importante tener en cuenta que el resumen, cuando se realiza en un operando, puede producir una salida que es igual al mismo operando. Según una forma de realización de la presente invención el resumen incluye una función unidireccional en el operando. Esto permite evitar también cualquier intento por terceros de encontrar y deducir las claves precedentes o instrucciones precedentes. Una función hash es un ejemplo de tal función unidireccional (SHA2, MD5 por ejemplo). El término "combinación" hace referencia a cualquier forma de combinación de los operandos mencionados, sean lógicos, aritméticos o criptográficos. De esta manera se garantiza el flujo y el contenido del programa ya que, si la instrucción corriente cifrada no es la instrucción prevista por el autor del programa, entonces la clave de descifrado corriente (KC), cuando se usa para descifrar la instrucción cifrada corriente, producirá otros valores no previstos. Obtenemos así una unidad de software que realiza las verificaciones por sí misma ya que la integridad del software se garantiza simplemente en virtud de su ejecución exitosa. Si el software ha sido manipulado de manera fraudulenta, entonces no se podrá ejecutar.

[0016] La Fig. 2 muestra un diagrama de flujo que representa la forma de realización anterior de la presente invención. Esta representación describe la invención desde el punto de vista de una instantánea, en vez de hablar de una instrucción corriente con su clave de descifrado corriente y de una instrucción precedente con su clave precedente, etc., se refiere sólo a una clave de instrucción (KI) que se actualiza a medida que se ejecuta cada instrucción. Como es normal en cualquier unidad de tratamiento, un contador de programa (PC) se utiliza para indicar la ubicación de la próxima instrucción a ejecutar. El contador de programa aumenta después de la ejecución de una instrucción o al contrario se actualiza si dicha instrucción dicta otra forma diferente de actualización que un simple aumento. Por ejemplo, si una instrucción implica una orden de carga de un valor a partir de un registro, entonces el contador de programa aumentará simplemente en general para indicar la posición siguiente. No obstante, si la instrucción implica un salto en una posición determinada, entonces el contador de programa será actualizado con el valor de la posición indicado por el salto.

[0017] El contador de programa (PC) y la clave de instrucción (KI) se inicializan primero (INI PC, INI KI). Una instrucción cifrada es leída desde la memoria del programa en un sitio indicado por el contador de programa (RD INST' c.f. PC) y descifrada mediante el uso de la clave de instrucción (DCPT INST, KI). La instrucción es ejecutada (EX INST) y el contador de programa es actualizado (UPD PC) por un simple aumento o bien por substitución de un nuevo valor como lo dicta la instrucción. La clave de instrucción es actualizada (UPD KI, INST) mediante el uso de un resumen de la instrucción ejecutada. La actualización de la clave de instrucción por lo tanto tiene en cuenta no sólo la instrucción que acaba de ser ejecutada sino también el valor de la clave utilizada para descifrar la instrucción. Sucesivamente, la clave de instrucción que se usó previamente para descifrar la instrucción precedente se construyó a partir de la instrucción precedente y la clave de instrucción que se usó para descifrar la instrucción antes de esto. De esta manera el valor de la clave de instrucción no sólo depende de la última instrucción ejecutada sino de todas las instrucciones combinadas ejecutadas previamente. De hecho, en una forma de realización de la presente invención, la actualización de la clave de instrucción tiene en cuenta el valor de la última instrucción ejecutada y los valores de al menos las dos instrucciones precedentes ejecutadas. Por ejemplo, la clave para descifrar la instrucción 4 podría ser una combinación de un resumen de instrucción 3, un resumen de instrucción 2 y un resumen de instrucción 1.

[0018] Como mostrado en la Fig. 2, el método de la presente invención implica un bucle donde la clave de instrucción se actualiza mediante el uso de la instrucción ejecutada previamente. Esto lleva a la pregunta de saber cómo descifrar la

primera instrucción en un programa. Si no hay ninguna instrucción ejecutada previamente, entonces ¿Cómo se calcula la primera clave de instrucción? En una forma de realización de la presente invención, la primera instrucción en un programa se deja en claro mientras que todas las otras instrucciones son cifradas. La primera instrucción por lo tanto se ejecuta directamente, de manera a iniciar el bucle, y la segunda instrucción se descifra mediante el uso de una clave de instrucción basada en la primera instrucción, y así sucesivamente. En otra forma de realización de la presente invención el programa entero es cifrado, incluyendo la primera instrucción, y la clave de instrucción se inicializa usando un valor que descifrará la primera instrucción. Este valor podría ser una clave maestra que se construye en el módulo de seguridad o de otra manera se comunica al módulo de seguridad desde el exterior.

[0019] Durante la ejecución de un programa, pueden surgir situaciones mediante las cuales una instrucción corriente (INSTC'), que reside en un sitio de memoria corriente (C) puede ser indicada como referencia por más de una de las instrucciones precedentes (INSTP1, INSTP2). En otras palabras una instrucción corriente, o receptor de llamada, se puede indicar como referencia por más de un solicitante, por ejemplo cuando se encuentra con una instrucción de tipo rama (incluyendo salto, rama o llamada por ejemplo). La Fig. 3 ilustra un escenario donde dos solicitantes (INSTP1 INSTP2) se refieren a un receptor de llamada (INSTC). En este caso, como puede haber dos valores distintos para la clave de instrucción en vista de diferentes historias posibles, esto llevaría a dos resultados diferentes dependiendo de las claves utilizadas para descifrar la función cifrada. Por supuesto ésta no es una situación deseable ya que la función cifrada sólo puede ser cifrada por una clave. Para evitar este problema, una modificación (CORR1 CORR2) se efectúa en el cálculo para forzar la clave de instrucción resultante hasta el valor requerido para descifrar correctamente la función. Por ejemplo, una función residente en el sitio C se indica en referencia por dos solicitantes diferentes residentes en los sitios P1 y P2. La clave de instrucción requerida para descifrar correctamente la instrucción cifrada en el sitio C (INSTC') es KC_{IN}. No obstante, el valor de la clave de instrucción después de la ejecución del residente de instrucción en P1 (INSTP1) es $KP1_{OUT}$ y el valor de la clave de instrucción después de la ejecución del residente de instrucción en P2 (INSTP2) es $KP2_{OUT}$. Además, se debe tener en cuenta que $KP1_{OUT}$ no es igual a $KP2_{OUT}$ y que ni $KP1_{OUT}$ ni $KP2_{OUT}$ son iguales a KC_{IN}. El método por lo tanto requiere que se aplique una modificación (CORR1 CORR2) con la que se puede así llevar el valor de clave de instrucción hasta el valor necesario siempre que se ejecute una instrucción de tipo rama. Como se conoce el valor de la clave requerido para descifrar el receptor de llamada (a saber, KC_{IN}) y que se conoce el valor de la clave después de la ejecución del solicitante, se puede predecir un valor de modificación para cada solicitante, donde el valor de modificación, cuando se usa en el cálculo, llevará la clave de instrucción hacia el valor requerido. El valor de modificación apropiado se aplica entonces en cada tipo rama para efectuar la modificación necesaria en la clave de instrucción cada vez que se utiliza este tipo de instrucción - una modificación diferente siendo hecha por el solicitante. Según una forma de realización de la presente invención el valor de modificación se introduce en forma de otro operando en la combinación de la clave de descifrado precedente y el resumen de la instrucción precedente tal y como se ha descrito anteriormente.

[0020] Como un ejemplo de cómo se efectúa la modificación sobre una clave de instrucción descrita anteriormente, consideraremos una instrucción de salto. En una forma de realización preferida de la presente invención, una instrucción de salto comprende un parámetro de destino, como es normalmente el caso para una instrucción de salto, y comprende también un parámetro de modificación, por ejemplo JMP C, #CORR1. El valor de modificación (#CORR1) se usa después como un parámetro adicional en la combinación de la clave de instrucción precedente y en toda o en una parte de la instrucción precedente. Es útil notar que en vez de extraer el valor de modificación de la instrucción y de utilizarlo como un parámetro extra en la fase de combinación, el resumen de la instrucción de salto puede tener en cuenta previamente el valor de modificación. La tabla siguiente T1 ilustra el estado de las claves de instrucción a medida que la ejecución de un programa ocurre a través de una instrucción de salto modificada del tipo descrito más arriba. La tabla incluye el valor de la clave requerida para descifrar una instrucción y el valor de la clave después de la ejecución de la instrucción y el cálculo de una nueva clave. Como se conoce el valor de la clave requerido para descifrar la instrucción en la etiqueta1, se deduce que los valores de corrección apropiados, CORR1 o CORR2, se pueden calcular para que los valores no modificados, K4 o K14, tengan el valor requerido K91.

T1

Clave requerida	Etiqueta	Instrucción	Clave resultante
K1		Instrucción 1	K2
K2		Instrucción 2	K3
K3		JMP label1, CORR1	K91=Fn(K4,CORR1)
K11		Instrucción 11	K12
K12		Instrucción 12	K13
K13		JMP etiqueta1, CORR2	K91=Fn(K14,CORR2)
K91	etiqueta1	Instrucción 91	K92
K92		Instrucción 92	K93

[0021] En otra forma de realización de la presente invención, en vez de con una instrucción de salto modificada por ejemplo, una instrucción de salto estándar se usa y la modificación hecha a la clave de instrucción descrita anteriormente se realiza por una instrucción dedicada "modificación" con un valor de modificación como parámetro. La función de tal instrucción de modificación consiste en actuar directamente sobre la clave de instrucción basada en el valor de modificación. La instrucción de modificación se sitúa justo antes de la derivación o de la instrucción de tipo salto, permitiendo así que la clave de instrucción sea actualizada de manera apropiadamente para descifrar correctamente el receptor de llamada. Es útil notar que la función de "modificación" como se ha descrito anteriormente se puede definir en realidad como una pluralidad de instrucciones diseñadas para ejecutar la operación de modificación deseada según el valor de la clave de instrucción. Por ejemplo, si el valor de la clave de instrucción requerido para descifrar correctamente el receptor de llamada es #39, entonces justo antes de citar una instrucción de salto, puede haber un XOR de la clave de instrucción (KI) con #39 para descubrir el valor de modificación (CORR1) y luego una adición de CORR y KI para producir un nuevo valor KI (corregido):

[0022] A modo de otro ejemplo, la clave de instrucción del receptor "etiqueta1" tiene el valor K91. Debido al hecho de que el flujo de programa puede llegar de diferentes trayectorias, una instrucción de corrección Inst_CORR se añade justo antes del salto de modo que la clave de instrucción se actualiza en un valor predeterminado K90. La ejecución de la instrucción de tipo rama, que en este caso es un salto, modificará la clave de instrucción de K90 a K91. Como es aparente en la tabla T2 más abajo, el valor de corrección (C1 C2) asociada a la instrucción de corrección (Inst_CORR) tiene como objetivo modificar la clave de instrucción corriente (K3, K13) hasta el valor predefinido K90. En consecuencia, la ejecución del salto actualizará la clave de instrucción de K90 a K91, el valor usado para descifrar la instrucción en el receptor (etiqueta1).

[0023] En caso de que la instrucción de tipo rama tenga un valor diferente, por ejemplo cuando la instrucción es una rama corta BRA, el resumen producido por esta instrucción será diferente al resumen producido por la instrucción de salto. En consecuencia, el valor de corrección C3 unido a la instrucción de corrección Inst_CORR debería tendrá en cuenta la diferencia, y la clave de instrucción durante la ejecución de la instrucción de rama no será el mismo que la instrucción de salto. No obstante, debido al valor de corrección C3, el valor final después la ejecución de la instrucción de rama corta seguirá siendo K91.

T2

Clave requerida	Etiqueta	Instrucción	Clave resultante
K1		Instrucción 1	K2
K2		Instrucción 2	K3
K3		Inst_CORR, C1	K90=Fn(C1,K4)
K90		JMP etiqueta1	K91
K11		Instrucción 11	K12
K12		Instrucción 12	K13
K13		Inst_CORR, C2	K90=Fn(C2,K14)
K90		JMP etiqueta1	K91
K20		Instrucción 1	K2
K21		Instrucción 2	K3
K22		Inst_CORR, C3	K80=Fn(C3,K23)
K80		BRA etiqueta1	K91
K91	label1	Instrucción 91	K92
K92		Instrucción 92	K93

[0024] La tabla siguiente T3 ilustra el estado de las claves de instrucción a medida que la ejecución de un programa evoluciona a través de una instrucción de salto condicional, donde dos destinos diferentes, etiqueta1 y etiqueta2, son posibles después de la ejecución del salto condicional. En este caso la clave requerida para descifrar las instrucciones en ambos destinos debería ser la misma. La tabla incluye el valor de la clave requerida para descifrar una instrucción y el valor de la clave después de la ejecución de la instrucción y el cálculo de una nueva clave.

T3

Clave requerida	Etiqueta	Instrucción	Clave resultante
K91	L1	Instrucción1	K2
K2		Instrucción2	K3
K3		CORR=C1	K90
K90		JMP COND L1, L2	K91
K11		Instrucción11	K12
K12		Instrucción12	K13
K13		JMP L2, CORR2	K91
K91	L2	Instrucción91	K92
K92		Instrucción92	K93

[0025] Otra situación donde un receptor de llamada puede ser indicado como referencia por una pluralidad de solicitantes, requiriendo así que se realice una modificación en la clave de instrucción para descifrar adecuadamente el receptor de llamada, es una llamada de función o una llamada de subprograma. Típicamente, durante este tipo de llamada, se pueden pasar unos parámetros durante la llamada de manera a aumentar el número de diferentes flujos posibles en la función o subprograma y en consecuencia el número de resultados posibles después de la ejecución de la función o subprograma. Cuando se produce tal llamada una modificación se realiza en la clave de instrucción de modo que su estado se puede conocer al inicio de la función o subprograma y otra modificación se realiza al regreso de la llamada es decir justo antes de salir de la función o subprograma.

[0026] Es importante indicar que en el contexto de la presente invención, la modificación como se ha descrito anteriormente también puede suponer simplemente una sustitución de una clave por otra clave.

[0027] Como lo saben ya los expertos en la técnica del tratamiento de datos, una instrucción comprende al menos un opcode, el cual define una operación que se debe realizar. La instrucción puede comprender sólo este último o bien comprender también uno o una pluralidad de operandos sobre los cuales se debe realizar la operación. Además del opcode y del operando u operandos si éstos existen, una instrucción puede comprender una marca de autentificación, también conocida como una figura de integridad, que se usa como una forma de control de la validez de la instrucción. Consecuentemente, en otra forma de realización de la presente invención, antes de la ejecución de una instrucción, la instrucción puede se puede comprobar primero mediante el uso de una marca de autentificación tal como se ha descrito anteriormente. La marca de autentificación puede tener la forma de una suma de comprobación o un valor hash de todo o de parte del opcode y operando(s). En la mayoría de los casos, la marca de autentificación se puede considerar como una firma del opcode. En el cifrado de toda o de una parte de una instrucción, nos encontramos así frente al hecho de saber si se debe elegir cifrar sólo el opcode o el opcode con la marca de autentificación o si se debe incluir también el(los) operando(s). Cualquier combinación tal como citada anteriormente, no obstante, en algunos casos en los que es importante ocultar el contenido de una programa a terceros, la presente invención favorece el cifrado del opcode y de la marca de autentificación, ya que la marca de autentificación puede proporcionar a un atacante potencial un indicio sobre el opcode. El método usado en esta forma de realización de la presente invención comprende así la lectura de una instrucción corriente cifrada; el uso de la clave de instrucción para descifrar la instrucción corriente cifrada y la marca de autentificación; la verificación la marca de autentificación extraída de este modo; la actualización de la clave de instrucción mediante el uso de un cálculo basado en el valor corriente de la clave de instrucción (o un resumen del mismo) y un resumen de la instrucción corriente, de tal modo que el siguiente instrucción cifrada a leer se puede descifrar con la clave de instrucción actualizada; la ejecución de la instrucción corriente con la condición de que la marca de autentificación sea válida. Si la marca de autentificación no es válida, entonces el programa puede estar concebido para terminarse fácilmente, es decir generando una alarma apropiada.

[0028] Puesto que en algunos casos donde la presente invención se puede implementar, un objetivo no debe consistir necesariamente en impedir que un tercero pueda copiar una unidad de software sino simplemente impedir que un tercero altere el software sin que esta alteración sea detectada, una forma de realización de la invención existe en el lugar donde los opcodes de las instrucciones se dejan en lenguaje claro y sólo las marcas de autentificación están cifradas. Esto es suficiente para conseguir el objetivo de garantizar la integridad de un software provisto por la invención. De forma similar, en otra forma de realización se puede cifrar sólo los operandos susceptibles de existir. Asimismo, se puede obtener el cifrado de cualesquiera opcode, operandos o marcas de autentificación o de cualquier otra combinación de éstos.

[0029] De forma similar, en una forma de realización de la presente invención, se puede mantener el opcode y los operandos en lenguaje claro y cifrar sólo parte de la marca de autentificación. Así en el caso de un salto, en vez de usar un valor de modificación como se ha descrito anteriormente, se puede desactivar simplemente el control de la marca de autentificación después de una instrucción de salto. La ventaja de esta solución es que la instrucción de salto no requerirá entonces un valor de modificación.

[0030] La presente invención por lo tanto provee una solución al problema de garantizar la integridad de programas de software mediante el cifrado de toda o de una parte de cada instrucción de una programa mediante el uso de una clave basada en toda o en una parte de una o de una pluralidad de instrucciones precedentes, produciendo así como resultado una clave de cifrado distinta por instrucción. La invención es aplicable a programas de software cuyas estructuras no son necesariamente de naturaleza arborescente y también se puede aplicar cuando el programa incluye bucles, saltos, llamadas o pausas etc. la invención permite señalar una excepción cuando se ha descifrado una instrucción cifrada de manera incorrecta. La primera instrucción no debe necesariamente estar en lenguaje claro ya que la clave de instrucción puede ser inicializada de forma apropiada como se requiere. La invención se puede realizar en el software o enteramente en el hardware de manera a eliminar así la posibilidad para un tercero de interceptar una instrucción descifrada o una clave de descifrado.

[0031] El cifrado de la instrucción puede emplear uno de una gran serie de algoritmos de cifrado tal como un cifrado de flujo, un cifrado de bloque, una libreta de un solo uso, un aleatorizador tal como una inversión de bits, un desplazamiento de bits, un intercambio de bits, un algoritmo de paridad o un código de redundancia cíclica por ejemplo.

REIVINDICACIONES

1. Método de implementación de un procesador para garantizar la integridad de un software en una memoria de programa, dicho software comprendiendo una pluralidad de instrucciones cifradas, una instrucción comprendiendo al menos un opcode, dicho método usando una clave de instrucción inicializada y comprendiendo los etapas siguientes de:

   - lectura de una instrucción corriente cifrada,

   - uso de la clave de instrucción para descifrar al menos parte de la instrucción corriente cifrada,

   - actualización de la clave de instrucción usando un cálculo basado en un resumen del valor corriente de la clave de instrucción y un resumen de la instrucción corriente, de tal modo que la siguiente instrucción cifrada a leer se puede descifrar con la clave de instrucción actualizada,

   - ejecución de la instrucción corriente.

2. Método según la reivindicación 1, donde una primera instrucción en la memoria de programa no está cifrada.

3. Método según la reivindicación 1 o 2, donde la instrucción corriente comprende también una marca de autentificación, dicha marca de autentificación siendo usada para autenticar dicha instrucción antes de la ejecución.

4. Método según cualquiera de las reivindicaciones 1 a 3, donde una modificación se aplica a la clave de instrucción, dicha modificación permitiendo el descifrado de la siguiente instrucción cifrada por medio de dicha clave de instrucción modificada para proporcionar una instrucción ejecutable.

5. Método según la reivindicación 4, donde la instrucción corriente comprende además un valor de modificación que se debe utilizar para realizar la modificación, dicho valor de modificación siendo extraído del valor de instrucción y actuando sobre la fase de actualización mientras que se determina la siguiente clave de cifrado.

6. Método según cualquiera de las reivindicaciones 1 a 5, donde cualquiera o todos los procesos de descifrado de una instrucción cifrada, de actualización de la clave de instrucción, de autenticación de la instrucción corriente o de la ejecución de la instrucción corriente se realizan dentro de un módulo de seguridad.

7. Método según cualquiera de las reivindicaciones 1 a 6, donde dicho resumen es un resultado de una función aplicada a toda o a una parte de dicha instrucción corriente, dicha función siendo seleccionada en el grupo que consiste en una función lógica, función aritmética, función criptográfica o función unidireccional.

8. Método según cualquiera de las reivindicaciones 1 a 7, donde la actualización de la clave de instrucción se basa posteriormente en un valor de modificación, dicho valor de modificación siendo usado para otorgar a la clave de instrucción un valor conocido.

9. Método según cualquiera de las reivindicaciones 1 a 8, donde se utiliza una clave maestra para inicializar la clave de instrucción.

10. Dispositivo comprendiendo un contador de programa (PC) y una memoria de programa (PMEM) para memorizar un programa cifrado, dicho programa cifrado comprendiendo una pluralidad de instrucciones cifradas (INST'), dichas instrucciones comprendiendo al menos un opcode, dicho dispositivo comprendiendo además un módulo de descifrado (DECR) y una unidad de tratamiento de datos (SCPU), dicho dispositivo teniendo acceso a una clave de instrucción inicializada (KI), donde dicho dispositivo se **caracteriza por el hecho de** comprender también medios para actualizar recursivamente la clave de instrucción (KI) en base a toda o a una parte de dicha clave de instrucción y un resumen de al menos una instrucción ejecutada previamente.

11. Dispositivo según la reivindicación 10, donde los medios para actualizar recursivamente la clave de instrucción se realizan en el hardware.

12. Dispositivo según cualquiera de las reivindicaciones 10 u 11, donde la actualización de la clave de instrucción se basa también en un valor de modificación, dicho valor de modificación siendo utilizado para otorgar a la clave de instrucción un valor conocido.

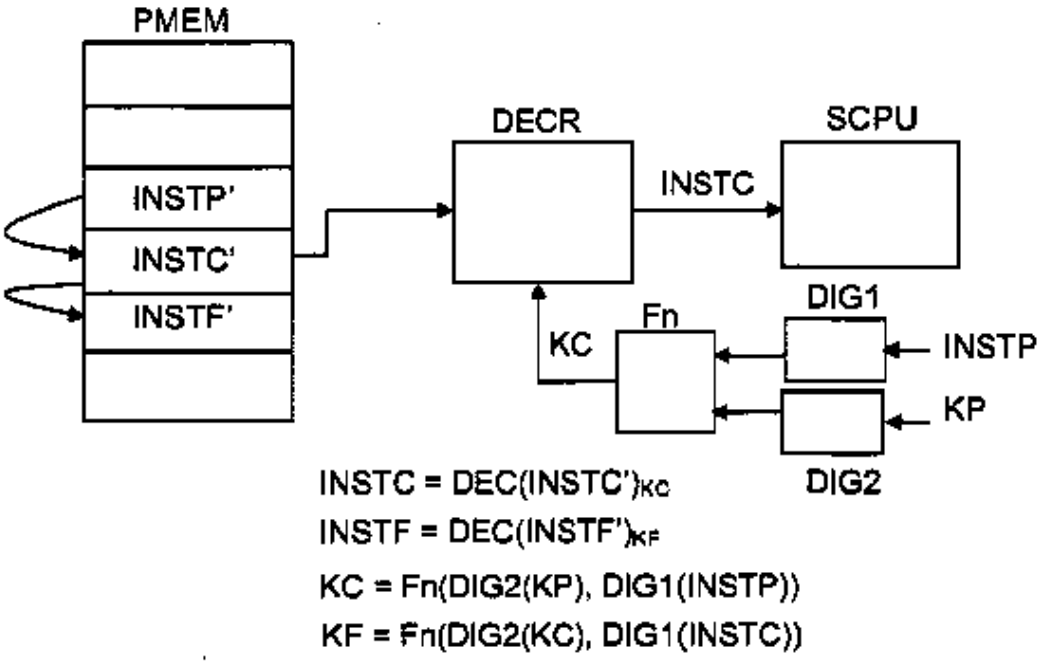
PMEM
INSTP'
INSTC'
INSTF'
DECR
INSTC
SCPU
DIG1
Fn
KC
INSTP
KP
DIG2
INSTC = DEC(INSTC')KC
INSTF = DEC(INSTF')KF
KC = Fn(DIG2(KP), DIG1(INSTP))
KF = Fn(DIG2(KC), DIG1(INSTC))

Fig. 1

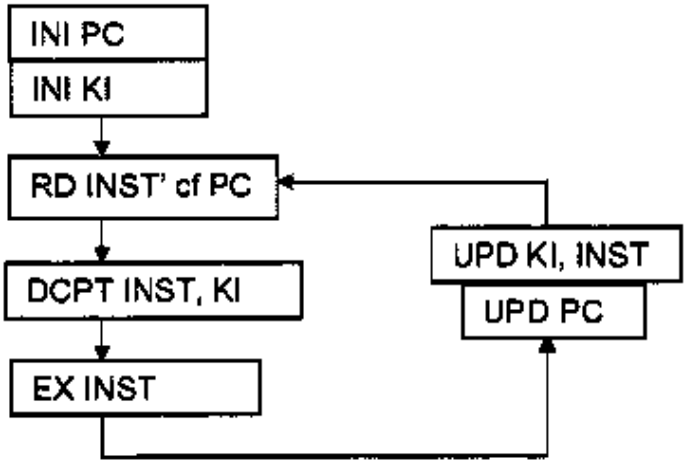
INI PC
INI KI
RD INST' cf PC
DCPT INST, KI
EX INST
UPD KI, INST
UPD PC

Fig. 2

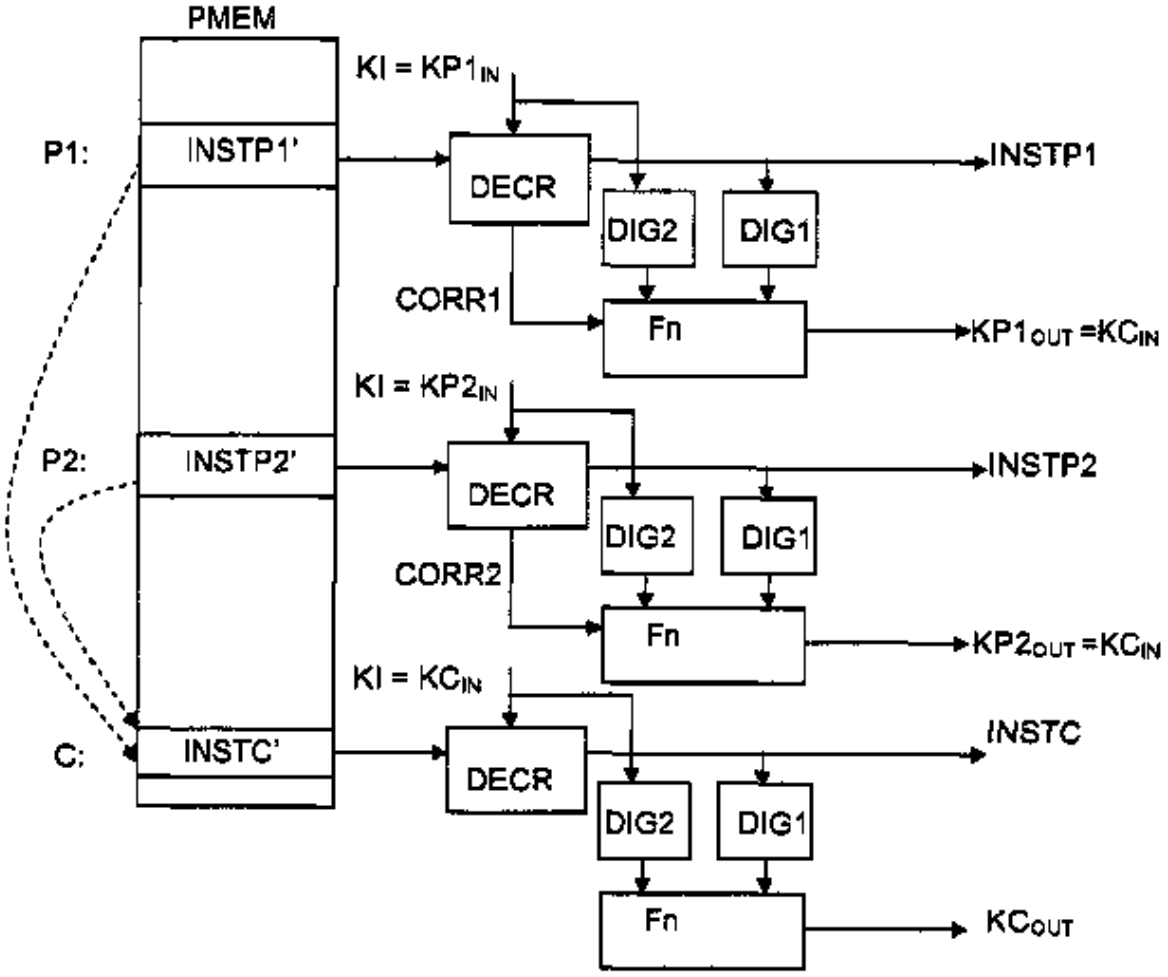
PMEM
KI = KP1IN
P1:
INSTP1'
DECR
INSTP1
DIG2
DIG1
CORR1
Fn
KP1OUT =KCIN
KI = KP2IN
P2:
INSTP2'
DECR
INSTP2
DIG2
DIG1
CORR2
Fn
KP2OUT =KCIN
KI = KCIN
C:
INSTC'
INSTC
DECR
DIG2
DIG1
Fn
KCOUT

Fig. 3