(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 664 020**

(51) Int. CI.:

G06F 9/445 (2006.01)
G06F 9/455 (2006.01)
G06F 9/44 (2006.01)
G06F 21/54 (2013.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **18.07.2007 PCT/EP2007/057417**

(87) Fecha y número de publicación internacional: **24.01.2008 WO08009697**

(96) Fecha de presentación y número de la solicitud europea: **18.07.2007 E 07787678 (7)**

(97) Fecha y número de publicación de la concesión europea: **25.10.2017 EP 2047366**

(54) Título: **Un método de protección dinámica de los datos durante la ejecución de un código de Software en lenguaje intermedio en un aparato digital**

(30) Prioridad:

20.07.2006 EP 06291183

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
18.04.2018

(73) Titular/es:

GEMALTO SA (100.0%)
6, rue de la Verrerie
92190 Meudon, FR

(72) Inventor/es:

GONZALVO, BENOIT y
FOURNIER, JACQUES JEAN-ALAIN

(74) Agente/Representante:

CASANOVAS CASSA, Buenaventura

DESCRIPCION

Un método de protección dinámica de los datos durante la ejecución de un código de software en lenguaje intermedio en un aparato digital

La presente invención se refiere a software y entornos de ejecución cargados en un aparato digital y, en particular, un método de protección dinámica de datos, en particular en vista de los ataques debidos a un fallo, durante la ejecución de una aplicación compilada en lenguaje intermedio en un aparato digital equipado con una máquina de ejecución virtual.

La invención se aplica a cualquier tipo de aparato digital, portátil o no, como un ordenador, pero también cualquier aparato equipado con un microcontrolador que comprende un procesador y dispositivos de almacenamiento, como una tarjeta inteligente.

El deseo de crear aplicaciones inter-operables ha dado como resultado el desarrollo de lenguajes de programación intermedios. El objetivo principal de dichos lenguajes es, pues, hacer su software independiente del hardware en el que se deben ejecutar. Por consiguiente, el software está previsto para ser ejecutado en forma de un código independiente intermedio de la arquitectura subyacente.

Los programadores son así universalmente liberados de las limitaciones relacionadas con un hardware específico. Los lenguajes intermedios como el código de bytes de Java, obtenido después de la compilación del lenguaje fuente de Java, han sufrido, por tanto, un desarrollo considerable. El lenguaje intermedio MSIL (acrónimo de "Lenguaje Intermedio de Microsoft") también puede ser citado: se despliega en el marco del entorno .Net o DotNet (marca registrada), obtenido tras la compilación de varios lenguajes de origen posibles, tales como C++ o C #.

El código intermedio corresponde por lo tanto de forma convencional a una forma compilada del software. Dicho software, compilado en Java, o en otros lenguajes intermedios, tales como .Net, no puede ser ejecutado como tal por el procesador del aparato en el que se desea ejecutar un programa compilado en forma de código intermedio. Es necesario introducir una capa de software que tenga por función principal de interpretar el código intermedio en las instrucciones que pueden ser ejecutadas por el procesador del aparato huésped. Dicha capa de software se conoce como una "máquina virtual". Por ejemplo, la máquina virtual JAVA permite que el software de Java se ejecute en una plataforma determinada en la que es implementada.

El software de Java se distribuye de forma convencional en la forma de un conjunto de módulos que consisten en archivos .class, que corresponde a una forma compilada del software. Cada archivo compilado corresponde a una estructura de datos de tipo class y comprende, en la medida de lo posible, la información para dicha clase: es decir, la descripción de los elementos de la clase (sus constantes, sus campos, sus métodos), la descripción de los elementos utilizados por la clase y definidos en otras clases (campos y métodos), el código de los métodos de la clase en la forma de instrucciones que puede ser interpretada (bytecode) por el intérprete de la máquina virtual Java.

De manera simplificada, un archivo .class tiene por ejemplo la siguiente estructura:

```
    Archivo de clase {
    Lista de campo; // Descripción de los campos de
la clase
    Lista de Métodos; // Métodos de dicha clase
(incluyendo su bytecode, es decir, que las instrucciones que
pueden ser interpretados por el intérprete de la máquina
virtual)
    }
```

Por lo tanto, a modo de ejemplo, en el marco de una aplicación de monedero electrónico Java, una clase conocida como "Monedero" se puede definir, con su campo "equilibrio" y su método "decrementBalance()". El archivo .class puede tener la siguiente estructura:

```
    Monedero clase pública  {
    balance int privado = 0;
    anular público decrementBalance () {
        este.balance = este.balance - 1;
          }
}
```

Por lo tanto, de acuerdo con dicho ejemplo, la ejecución del método de la clase "Monedero" consiste en eliminar el valor 1 en la instancia en curso del campo de equilibrio.

La ejecución del correspondiente software de Java se realiza por medio de la máquina virtual Java instalada en el

aparato digital.

Dicha máquina virtual transforma la información del archivo de clase en las estructuras de almacenamiento de datos de trabajo que son específicos de la máquina virtual y que permiten a dicha máquina virtual interpretar y ejecutar el software. Dichas estructuras de datos se organizan convencionalmente en la forma de pilas, en las que los datos se almacenan uno encima del otro en el orden en el que llegan.

La máquina virtual tiene un conjunto de instrucciones específicas, estando codificada cada instrucción en uno o más octetos. El conjunto de instrucciones de la máquina virtual comprende por ejemplo un cierto número de instrucciones convencionales, tales como la aritmética, operaciones lógicas y saltos.

Actualmente, la ejecución de una aplicación por parte de la máquina virtual no está asegurada por completo, en particular en relación con los ataques en general y, en particular, con los ataques debidos a un fallo. Así, el valor sensible en el ejemplo antes mencionado, que se muestra en el campo "balance", puede ser modificado después de la inyección de un fallo durante la manipulación de dicho valor para la interpretación del método para la actualización de dicho campo por la máquina virtual, que resulta en establecer un valor final en la pila de ejecución para el campo de equilibrio diferente al esperado normalmente. Por ejemplo, el campo de equilibrio puede ser forzado a su valor máximo.

Una manera conocida de combatir dicho tipo de ataques debido a un fallo consiste en añadir un dispositivo de protección a nivel de la aplicación, en forma de un código de control redundante insertado dentro mismo del código de la aplicación que ha de ser ejecutado por la máquina virtual.

Por ejemplo, con el fin de proteger el código precedente, es posible utilizar en Java, como una suma de comprobación, el valor complementado del campo de equilibrio, comprobado ~ equilibrio, que envía el complemento a 1 del valor de equilibrio binario, y actualiza dicho valor de suma de comprobación al mismo tiempo con la actualización del valor de equilibrio durante la ejecución del método por la máquina virtual. La comparación del resultado de los dos cálculos de actualización realizada al mismo tiempo, por un lado, para el campo de equilibrio y, por el otro, el completado ~ campo de equilibrio, permite por tanto la integridad de los datos a verificar que se utilizaron para establecer los cálculos. El código Java modificado en consecuencia por lo tanto, tiene la siguiente estructura:

```
    Mondero clase pública  {
    balance int privado  = 0;
    balChecsum int privado = ~balance;  // = OxFFFFFFFF
    anular público decrementBalance () {
        este.balance = este.balance  - 1;
        este.balChecsum = este.balChecsum + 1;
          }
}
```

La desventaja de dicho método para asegurar la ejecución del código Java es que consume gran cantidad de recursos y tiempo de cálculo, lo que es desfavorable en entornos restrictivos tales como el de las tarjetas inteligentes.

Además, dicha solución implica que los desarrolladores de código estén familiarizados y sean conscientes, en el momento del desarrollo, de la manera de proteger las partes sensibles del código. Por lo tanto, en realidad resulta restrictivo para los desarrolladores añadir códigos con el fin de proteger su aplicación.

El documento WO 01/88705 describe un método de verificación de los tipos de datos que integran la pila. La verificación realizada de manera dinámica durante la ejecución de un programa. Este método verifica la consistencia del tipo para los datos que son gestionados durante la ejecución del programa.

El documento WO 2005/101725 muestra un método de verificación de un programa mientras éste es ejecutado. El programa contiene una secuencia de instrucciones datos estáticos ejecutables. El método se basa en la comprobación de un hash correspondiente al programa y en una pluralidad de datos basados en las firmas de las instrucciones.

La invención pretende resolver uno o más de dichos inconvenientes.

El objeto de la invención es por lo tanto un método para proteger los datos de una aplicación compilados en código intermedio previsto para ser ejecutado en un aparato digital equipado con una máquina virtual que gestiona la ejecución del código a través de una pila de ejecución que define al menos un marco de pila correspondiente a un método convocado durante la ejecución del código.

Según la invención, el método incluye la aplicación de un modo de ejecución segura del código, que comprende:

- la determinación de al menos una suma de comprobación global asociada con cada marco de pila y,
- cada vez que se manipula un dato del código, el cálculo de una suma de comprobación asociado con dicho dato, basado por un lado, en la suma de comprobación global asociada con el marco de pila correspondiente al dato manipulado y, por otro, en al menos una parte de los otros datos que constituyen dicho marco de pila.

De acuerdo con una realización particular, el método comprende la división de cada marco de pila en una pluralidad de áreas y la determinación de una pluralidad de sumas de comprobación globales (gck0, gck1, gck2, gck3) asociadas respectivamente con cada área del marco de pila.

De acuerdo con dicha realización particular, la suma de comprobación local asociada con el dato manipulado comprende la determinación de antemano de la suma de comprobación global asociada con el área del marco de pila en la que se almacena el dato manipulado.

Ventajosamente, el cálculo de la suma de comprobación local asociada con dato manipulado se realiza por una parte, sobre la base de la suma de comprobación global asociada con el área del marco de pila en la que el dato manipulado se almacena y, por otra parte, en otros datos de dicha área del marco de pila.

Preferiblemente, cada marco de pila se divide en dos áreas, que corresponden respectivamente a un área variable local del método convocado y un área de pila de operandos de dicho método.

De acuerdo con una realización, el método se implementa en forma de software.

De acuerdo con otra realización, el método se implementa en forma de hardware.

La invención también se refiere a una máquina virtual para ejecutar una aplicación compilada en código intermedio en un aparato digital, caracterizado porque es probable que se almacene en un dispositivo de almacenamiento no volátil del aparato y es probable que implemente el método según la invención.

La invención también se refiere a un aparato digital que comprende un dispositivo de almacenamiento no volátil almacenado en la memoria de la máquina virtual de acuerdo con la invención.

Otras características y ventajas de la invención especiales se harán evidentes durante la lectura de la descripción realizada a modo de ejemplo no limitativo y en relación con las siguientes figuras, en las que:

- la figura 1 ilustra la estructura de almacenamiento de un marco de pila, a nivel del cual actúa el mecanismo de protección de acuerdo con la presente invención;
- las figuras 2a y 2b ilustran un primer ejemplo de realización de la invención;
- las figuras 3a a 3d ilustran un segundo ejemplo de realización de la invención;
- las figuras 4 a 6 ilustran las mejoras de la invención, y
- la figura 7 ilustra un ejemplo de implementación de hardware del método de acuerdo con la presente invención.

Por tanto, la invención tiene por objeto proteger la ejecución de una aplicación compilado en código intermedio en un aparato digital equipado con una máquina de ejecución virtual para la interpretación del código de la aplicación.

El mecanismo de protección según la presente invención actúa, tal como se verá más adelante con mayor detalle, a nivel del marco de pila, mostrando un bloque de almacenamiento reservado, en la pila de ejecución de la máquina virtual, durante la ejecución de un método del código.

La Figura 1 ilustra la estructura de dicho marco de pila 10, que contiene típicamente los parámetros del método convocado, es decir, sus variables locales 11, su pila de operandos 13, junto con los datos del sistema 12, comprendiendo en particular, la dirección de retorno que permite encontrar el marco de pila precedente. De hecho, es común que un primer método, con el cual está asociado un primer marco de pila, requiere para su ejecución en sí convocar un segundo método, con el que se asocia un segundo marco de pila. En este caso, cuando el primer método es convocado, en un momento dado del progreso de su ejecución, se convoca el segundo método, que implica trabajar en el marco de pila asociado con dicho segundo método. Cuando las acciones realizadas en el marco de pila asociada con el segundo método son finalizadas, los datos del sistema y, más concretamente, la dirección de retorno, del marco de pila asociado con el segundo método se utilizan para encontrar el marco de pila precedente asociado con el primer método.

El principio general de la invención por lo tanto consiste en la adición de un mecanismo de seguridad a nivel de cada marco de pila de la máquina virtual, con el objetivo de ser capaz de garantizar en todo momento la integridad de los datos mantenidos en el marco de pila asociado durante la ejecución de un método.

Para esto, con cada marco de pila de la máquina virtual tal como se ilustra en la figura 1, se asocia primero una suma de comprobación global gck. El valor de suma de comprobación global de gck se calcula utilizando todos los datos del marco de pila, es decir, todos los elementos que constituyen la pila de operandos 13 y todas las variables

locales 11.

Entonces, cada vez que un dato de un marco de pila, es decir, un elemento de la pila de operandos 13 o variables locales 11, es desapilado para cualquier operación, se puede así calcular dinámicamente una suma de comprobación local asociada, utilizando por un lado, la suma de comprobación global asociada con el marco de pila en cuestión, calculado de antemano y mantenido en el marco de pila y, por el otro lado, los otros datos del marco de pila en cuestión.

Será conveniente, cada vez que se manipula un dato de un marco de pila por parte de la máquina virtual de la ejecución, actualizar en consecuencia la suma de comprobación global asociado con el marco de pila.

Los principios generales explicados anteriormente en los que se basa la invención serán ahora descritos en mayor detalle. Para esto, se proporcionan primero las definiciones que serán útiles en lo sucesivo para la descripción de las funciones de cálculo de las sumas de comprobación globales, asociadas con cada marco de pila, y las sumas de comprobación locales, susceptibles de ser asociadas con cada dato de un marco de pila basado en la suma de comprobación global asociada.

De hecho, para el propósito de la invención, es necesario definir dos tipos de funciones, uno de los cuales está previsto para actuar globalmente en un marco de pila, mientras que el otro permite que los cálculos ejecutados por la máquina virtual en uno o más datos sean monitoreados a nivel local, en particular, basado en la suma de comprobación global asociada con el marco de pila correspondiente a los datos en cuestión. Mediante el uso de una combinación de funciones tal, se verá que es, por tanto, posible garantizar la integridad de los datos manipulados por la máquina virtual en cualquier momento de la ejecución del código.

Cualquier V el espacio de los valores en los registros s largos. Por ejemplo, para los registros largos de 32 bits, se obtiene lo siguiente:

$$V = [-2147483648,\ 2147483647] = [-2^{31},\ 2^{31}-1]$$

La función de cálculo de suma de comprobación GCK global asociada con un marco de pila se define de la siguiente manera:

```
GCK: V x V x … x V ---------> V
     (S0, S1, …, Sn-1)|------> GCK(S0, …, Sn-1)
```

S0, S1, ..., Sn-1, siendo todos los datos que constituyen un marco de pila dado.

La función de cálculo de suma de comprobación LCK local asociada con el dato S0 se define de la siguiente manera:

```
LCK: V ----------> V
     S0 |--------> LCK(S0)
```

La función GCK debe tener el siguiente atributo:

Ya sea la función GCK y LCK tal como se ha definido anteriormente, consideraremos cualquier función F como, para cualquier conjunto de datos (S0, ..., Sn-1):

$$GCK(S0,\ S1,\ \ldots,\ Sn\text{-}1) = F(LCK(S0),\ \ldots,\ LCK(Sn\text{-}1))$$

A modo de ejemplo, mediante el uso del operador ~ complemento aplicado a la suma de todos de los elementos de un marco de pila dado, se define lo siguiente:

```
GCK: V x V x … x V ----------> V
     (S0, S1, …, Sn-1)|-------> ~(S0 + … + Sn-1)

LCK: V ------------> V
     S0 |---------> ~S0
```

O la función f definida por:

```
F: V x V x … x V     -----> V
   (X0, X1, …, Xn-1)|---> Sum(Xi, i=0,…,n-1) + n-1
```

Por lo tanto :

$$GCK(S0, S1, \ldots, Sn-1) = \sim(S0 + \ldots + Sn-1) = \sim S0 + \ldots + \sim Sn-1 + n-1 = F(\sim S0, \ldots, \sim Sn-1) = F(LCK(S0), \ldots, LCK(Sn-1)).$$

A partir de ahí, si el valor gck de la suma de comprobación global asociada con un marco de pila (gck = GCK(S0, S1, ..., Sn-1)) es conocido, resulta entonces sencillo calcular el LCK(Si) de la suma de comprobación local asociado con un dato Si del marco de pila accedido por la máquina virtual, basado en el valor gck asociado con el marco de pila en cuestión y el dato Sj restante del marco de pila (j≠i), mediante el uso de la siguiente operación:

$$LCK(Si) = \sim Si = GCK(S0, S1, \ldots, Sn-1) - (\sim S0 + \ldots + \sim Si-1 + \sim Si+1 + \ldots + \sim Sn-1 + n-1)$$

De acuerdo con el ejemplo, con el fin de determinar la suma de comprobación local asociada con un dato Si del marco de pila, la suma complementada de los otros datos que constituyen el marco de pila resulta así retirada del valor gck de suma de comprobación global.

Esto muestra que la de suma de comprobación local LCK(Si) asociado con el dato Si, se puede calcular sin usar explícitamente el valor del dato Si, sino sólo poniendo en juego la suma de comprobación global gck asociado con el marco de la pila correspondiente al dato Si y los otros datos del marco de pila.

Dicho atributo es particularmente ventajoso en vista de reforzar la seguridad de la ejecución de aplicaciones dentro del contexto de la invención. De hecho, incluso si un atacante es capaz de modificar el valor de un dato de un marco de pila en el momento en el que este último es leído por la máquina virtual, la suma de comprobación local asociada que no es calculada utilizando el valor del dato en cuestión, será posible detectar que el dato en cuestión ha sido modificado mediante la comparación con el valor procedente de la suma de comprobación.

Las figuras 2a y 2b permiten que este asunto pueda ser ilustrado con mayor precisión. Vamos a suponer que el contenido de un marco de pila 10 sea descrito por el apilamiento de células de la figura 2a, comprendiendo respectivamente, d, c, b y unos datos.

La suma de comprobación global gck asociado con dicho marco de pila se puede calcular como la suma complementada con los elementos del marco de pila, utilizando el ejemplo de la función GCK antes proporcionada basado en el operador ~ complemento aplicado a la suma de todos los elementos del marco de pila:

$$gck = \sim(a + b + c + d)$$

Supongamos ahora que el dato a se desapila desde la parte superior del marco de pila 10 como se ilustra en la figura 2b, y que en el momento de dicha operación, su valor se modifica de manera fraudulenta como a' mediante medios externos.

Mediante la aplicación de los atributos vinculados al operador, se ha obtenido lo siguiente:

$$gck = \sim a + \sim b + \sim c + \sim d + 3$$

Como ya se ha explicado, la suma de control local LCK(a) asociada con el dato a manipulado en curso puede por tanto ser calculado basado en la suma de comprobación gck global y los datos restantes en el marco de pila, de manera que se obtiene lo siguiente:

$$LCK(a) = gck - \sim(b + c + d) - 1 = \sim a$$

La suma de comprobación local LCK(a) puede, por ejemplo, calcularse si el sistema solicita una verificación del dato a manipulado. En este caso, el valor del dato manipulado se comparará con el valor del dato derivado de la suma de comprobación local ~a asociada. Como el dato a, en el ejemplo, se ha modificado en una a' tras un ataque externo, la comparación con el valor obtenido de ~a indica por tanto al sistema que ha ocurrido un error, permitiendo entonces que reaccione.

Dicha etapa de verificación de la integridad se puede implementar opcionalmente. Así, se puede contemplar aplazarla, para realizarla en momentos predeterminados o incluso aleatorios.

La máquina virtual es capaz también de garantizar la integridad de los datos en sí durante un cálculo. Dicho mecanismo se ilustra más específicamente en el ejemplo de las figuras 3a a 3d.

Vamos a suponer un marco de pila 10, constituido por el apilamiento de los datos b, a, c y d, como se ilustra en la figura 3a. La suma de comprobación global asociada con el marco de pila se calcula de la siguiente manera:

$$gck = \sim(a + b + c + d)$$

Vamos a suponer que la ejecución del método implica el cálculo d + c. El primer argumento de la adición constituido por el dato d es, por tanto, inicialmente desapilado y, utilizando los principios antes mencionados, se puede calcular su suma de comprobación LCK(d) local basado en la suma de comprobación gck global asociada con el marco de pila que corresponde a los datos restantes en el marco de pila:

$$LCK(d) = gck - \sim(b + a + c) - 1 = \sim d$$

Entonces, la suma de comprobación gck global asociado con el marco de pila es actualizada antes de continuar la operación de cálculo, con el fin de reflejar el nuevo estado del marco de pila, habiendo sido desapilado el dato d. Para ello, el valor de la suma de comprobación local asociada con el dato d de referencia es eliminado del valor en curso de la suma de comprobación global:

$$gck = gck - (\sim d) - 1 = \sim(a + b + c)$$

Por consiguiente, la operación de cálculo se continúa y, como se ilustra en la figura 3c, el segundo argumento de la adición constituido por el dato c ahora es desapilado. Todavía con el propósito de verificar la integridad de los datos utilizados en el cálculo en curso, la suma de comprobación LCK(c) local asociado con el dato c es calculada dinámicamente sobre la base de la suma de comprobación gck global previamente actualizada, a la que se le elimina la suma complementada de los datos restantes en el marco de pila:

$$LCK(c) = gck - \sim(b + a) - 1 = \sim c$$

De la misma forma que antes, la suma de comprobación global asociada con el marco de pila debe actualizarse para reflejar el nuevo estado del marco de pila, en el que el dato c ha sido desapilado. Por consiguiente, el valor de la suma de comprobación local asociada con el dato c es eliminado del valor en curso de la suma de comprobación global:

$$gck = gck - (\sim c) - 1 = \sim(a + b)$$

En dicha etapa, los siguientes cuatro datos son, por lo tanto, manipulados por la máquina virtual: c, ~c, d y ~d, lo que por tanto permite comprobar que los datos puestos en juego en el cálculo no se han modificado en el momento en que son leídos, de la misma manera como se ha explicado en referencia con el ejemplo de las figuras 2a y 2b.

Por consiguiente, se realiza la operación de adición (c+d). La integridad de dicha operación se puede verificar utilizando las sumas de comprobación LCK(c) y LCK(d) locales, asociadas respectivamente con los datos c y d. En efecto, la operación de comprobación ~(c+d) correspondiente se escribe: ~ c + ~ d + 1, de acuerdo con los atributos aritméticos del operador ~ complemento. Así, ventajosamente la suma de comprobación LCK(c+d) local asociada con el resultado de la operación (c+d) puede ser calculado, sin tener que usar los valores de los datos c y d directamente, sino sólo usando las sumas de comprobación LCK(c) y LCK(d) locales previamente calculadas y asociadas con los datos que intervienen en la operación. Además, como ya se ha visto, las sumas de comprobación LCK(c) y LCK(d) locales han sido ellas mismas calculadas sobre la base de la suma de comprobación gck global, sin tener que utilizar los valores de los datos c y d.

Por último, como se ilustra en la figura 3d, una vez que el cálculo se ha realizado, el resultado (c + d) se coloca en el marco de pila 10. Por consiguiente, la suma de comprobación global asociado con el marco de la pila todavía ha de ser actualizada, mediante el uso de la suma de comprobación local asociada con el resultado:

$$gck = gck+(\sim(c + d)+1) = \sim(a + b) + \sim(c + d) + 1$$

$$= \sim (a + b + c + d)$$

En los ejemplos mencionados anteriormente, se utilizó el operador ~ complemento para el cálculo de las sumas de comprobación locales. La invención puede, sin embargo, generalizarse utilizando la siguiente generalización del operador ~:

Cualquier O del conjunto V, con las siguientes operaciones convencionales aritméticas y lógicas (+,-,*,/,&,|).

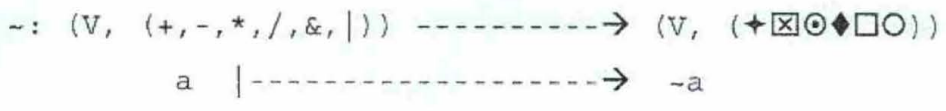
O la P del conjunto V con todas las operaciones simbolizadas de la siguiente manera (♦, , Θ,•,□,O).

Un operador de suma de comprobación Chk se define de la siguiente manera:

Chk: O---------- -> P, de tal manera que para cualquier operación op en el conjunto (+,-,*,/,&,|), hay una operación cop correspondiente en el conjunto (♦, , Θ,•,□,O) tal que:

$$Chk(a\ op\ b) = Chk(a)\ cop\ Chk(b)$$

Por consiguiente, el operador ~ previamente descrito del complemento a 1 es un caso específico de la función Chk:



con

$$\sim(a + b) = \sim a \spadesuit \sim b = \sim a + \sim b + 1$$

Matemáticamente, el operador Chk puede definirse como un morfismo grupo del espacio O hacia el espacio P, respetando todas las operaciones en P.

El principio de la invención, por tanto, consiste en asociar con cada marco de pila de la máquina de ejecución virtual una suma de comprobación global, cada vez que se accede a un dato, calcular una suma de comprobación local asociada con dicho dato mediante el uso de la suma de comprobación global asociada con el marco de pila en cuestión.

En los ejemplos de formas de realización proporcionadas anteriormente, el cálculo de la suma de comprobación global asociada con un marco de pila consiste en calcular la suma complementada de todos los elementos del marco de pila. Además, en modo seguro, cada vez que se accede a un dato del marco de pila, se calcula su suma de comprobación, mediante el uso de la suma de comprobación global a la que se resta la suma complementada de los otros elementos del marco de pila. También, si la longitud del marco de pila es sustancial, dichos cálculos pueden demostrar ser desfavorables en términos de actuaciones (ralentización de la ejecución del código en modo seguro) en entornos restringidos en los recursos de cálculo.

También, en el objetivo de reducir el número de operaciones necesarias para el cálculo de una suma de comprobación local basada en una suma de comprobación global o para el cálculo de la actualización de la suma de comprobación global, una mejora de la invención consiste en utilizar una pluralidad de sumas de comprobación globales asociados con el mismo marco de pila.

Según una primera variante ilustrada en la figura 4, se utilizan dos sumas de comprobación distintivas gck_{op} y gck_{v} globales para cada marco de pila 10, asociadas respectivamente con la pila de operandos 13 del marco de pila y con las variables locales 11 del marco de pila. Esto permite que los cálculos necesarios para la obtención de una suma de comprobación se vean consecuentemente reducidos. De hecho, si por ejemplo, un dato está manipulado desde la pila de operandos 13, se puede obtener su suma de comprobación local asociada de la suma de comprobación gck_{op} global asociada y la suma de los únicos otros elementos de la pila de operandos y no de todo el marco de pila.

Según otra variante presentada en la figura 5, el marco de pila se divide en una pluralidad de áreas, con cada uno de las cuales se asocia una suma de comprobación global. Las áreas que dividen el marco de pila son preferentemente de igual longitud, pero la división en áreas de extensiones desiguales puede, sin embargo, preverse. De acuerdo con el ejemplo de la figura 5, el área de las variables locales 11 se divide en dos áreas, cada una de cuatro elementos, con las sumas de comprobación gck0 y gck1 globales asociadas; asimismo el área de la pila de operandos se divide en dos zonas cada una de cuatro elementos, con respectivamente con las sumas de comprobación gck2 y gck3 globales asociadas.

De acuerdo con dicha forma de realización, cada vez que un dato de la pila es manipulado (leído o añadido), es primero necesario determinar a qué área del marco de pila se refiere, con el fin de utilizar la suma de comprobación global asociada con dicha zona para deducir la suma de comprobación local asociada con el dato manipulado.

Cada suma de comprobación global asociada con un área del marco de pila es, de hecho, identificada por un índice (0 a 3 de acuerdo con el ejemplo de la figura 5). Además, con el fin de determinar la suma de comprobación global a usar, es necesario determinar el índice que permite identificar la suma de comprobación global correcta. La determinación del índice se puede realizar de la siguiente manera:

$$index = (jSP - base)/n$$

En donde jSP representa el puntero sobre el elemento en curso del marco de pila, la base representa el puntero en la base del marco de pila y n representa el número de elementos de un área de longitud fijada del marco de pila.

En referencia al ejemplo de la figura 6 en el que se presume que el elemento es desapilado del marco de pila mostrado, dividido a modo de ejemplo en dos áreas de longitud n = 4, se obtiene el índice = 1, lo que significa que es necesario utilizar la suma de comprobación gck1 global para el área en cuestión.

El cálculo de la suma de comprobación local asociada con el dato a es, por tanto, realizado como se ha explicado anteriormente, mediante el uso de la suma de comprobación gck1 global determinada previamente y teniendo ventajosamente en cuenta solamente los datos del área afectada, en comparación con todos los datos del marco de pila.

El cálculo de una suma de comprobación local cumple la siguiente fórmula:

$$LCK(a) = gck(index) - \sum_{i=base+n*index}^{iSP} Si \ ,$$

Simbolizando Si los elementos del marco de pila.

Los cálculos de las sumas de comprobación locales y globales como han sido descritas pueden realizarse en una forma de software. Sin embargo, se ha visto que cuando los marcos de pila correspondientes a los métodos convocados durante la ejecución del código son de una longitud considerable, dichos cálculos pueden llegar a ser excesivamente largos. La variante antes mencionada, consistente en dividir el marco de pila en una pluralidad de zonas a cada una de los cuales se asociada una suma de comprobación global, permite ventajosamente la reducción del tiempo de cálculo.

Otra solución se basa en una implementación de hardware del cálculo de las sumas de comprobación de acuerdo con la invención. Un ejemplo de implementación de hardware se describe en la figura 7. De acuerdo con dicho ejemplo de implementación, un área de almacenamiento 20, que consiste en un conjunto de registros r1 a rn, está conectado a un circuito lógico 30, que permite que una suma de comprobación sea calculada en base al contenido de los registros n del área de almacenamiento 20 cada vez que uno de dichos registros es actualizado. Por tanto, el resultado del cálculo realizado por el circuito lógico 30 se coloca automáticamente en un registro de comprobación 40 como una suma de comprobación.

El circuito lógico 30 es, por ejemplo, conectado con el fin de ejecutar la suma de todos los registros seleccionados. La determinación automática de una suma de comprobación a través de la lógica cableada permite así que se incremente significativamente la rapidez de los cálculos en relación con una implementación puramente de software cuando el número n es alto.

El conjunto de registros r1 a rn se utiliza, por tanto, como un dispositivo de almacenamiento permitiendo que los datos correspondientes de un marco de pila que son asignados en el área de almacenamiento 20 sean apilados y desapilados. Un conjunto F de indicadores F1 a Fn está previsto además, en el que cada indicador F1 a Fn está asociado respectivamente con un registro r1 a rn, permitiendo una actualización del marco de pila a ser indicado. Por ejemplo, cuando un dato es colocado en la pila, el bit del indicador correspondiente se coloca a 1 y cuando un dato desapilado, el bit del indicador correspondiente se coloca a 0. Así, en cada cambio detectado en el conjunto de indicadores F, se recalcula la suma de comprobación a través del circuito lógico 30 basado en el contenido de la totalidad de los registros ri a rn de los que el bit del indicador correspondiente se coloca a 1.

El conjunto de hardware 20, 30, 40, 50 coopera con un dispositivo de interfaz 60 conectado a una unidad de procesamiento del componente en el que se ejecuta el código. Por tanto, se pueden realizar las operaciones de adición y eliminación de datos en un marco de pila, comprobadas a través de la configuración de hardware descrita.

Por lo tanto, el conjunto de datos de un marco de pila asignados en un área de almacenamiento 20, cuando un dato debe ser eliminado, la suma de comprobación en curso, conocida como CHK1, asociada con el marco de pila se lee primero en el registro 40. Entonces, una señal de "leer" indica al dispositivo de interfaz 60 que un dato es eliminado, éstos últimos comandan la actualización de los registros 20 y los indicadores 50 en la base de datos de las direcciones ADR y las señales de datos DAT recibidas por la unidad de procesamiento y un valor de suma de comprobación actualizado, conocido como CHK2, es calculado automáticamente y colocado en el registro de comprobación 40. Así, sobre la base de las sumas de comprobación CHK1 y CHK2 calculadas automáticamente, el sistema puede determinar la suma de comprobación local asociada con el dato eliminado con el fin de verificar su integridad.

De la misma manera, cuando un dato debe ser añadido a la pila, la suma de comprobación en curso, conocida como CHK1, asociada con el marco de pila es primeramente leía del registro 40. Una señal de "escribir" indica, por tanto, al dispositivo de interfaz 60 que se ha añadido un dato, comandado así la actualización de los registros 20 y los indicadores 50 en la base de datos de la dirección ADR y las señales de datos DAT recibidas por la unidad de procesamiento. Una vez que el dato se ha cargado en el registro correspondiente del área de almacenamiento 20, la suma de comprobación se calcula de nuevo de forma automática y el valor actualizado CHK2 se coloca en el registro de comprobación 40. Así, sobre la base de las sumas de comprobación CHK1 y CHK2 calculadas automáticamente, el sistema puede determinar la suma de comprobación local asociada con el dato añadido con el fin de verificar su integridad.

REIVINDICACIONES

1. Un método para proteger los datos de una aplicación compilada en código intermedio planificado para ser ejecutado en un aparato digital equipado con una máquina virtual que gestiona la ejecución del código a través de una pila de ejecución que define al menos un marco de pila (10) correspondiente a un método convocado durante la ejecución del código, **caracterizado porque** comprende la aplicación de un modo de ejecución seguro del código, que incluye:

- la determinación de al menos una suma de comprobación (gck) global asociada con un área de cada marco de pila y, calculándose dicha suma de comprobación (gck) global asociada utilizando el valor de todos los atos del área asociada,

- cada vez que un dato (a) del código es manipulado, el cálculo de una suma de comprobación (LCK(a)) asociada con dicho dato, basada por un lado, en la suma de comprobación global asociada con el área correspondiente al dato manipulado y, por otro lado, en al menos una parte de los otros datos que constituyen dicha área.

2. Un método de acuerdo con la reivindicación 1, **caracterizado porque** comprende la división de cada marco de pila (10) en una pluralidad de áreas y la determinación de una pluralidad de sumas de comprobación (gck0 gck1, gck2, gck3) globales asociadas respectivamente con cada área del marco de la pila.

3. Un método de acuerdo con la reivindicación 2, **caracterizado porque** el cálculo de la suma de comprobación local asociada con el dato manipulado comprende la determinación previa de la suma de comprobación global asociada con el área del marco de pila en el que se almacena el dato manipulado.

4. Un método de acuerdo con la reivindicación 3, **caracterizado porque** el cálculo de la suma de comprobación local asociada con el dato manipulado se realiza por un lado, en base a la suma de comprobación global asociada con el área del marco de pila donde se almacena el dato manipulado y, por otro lado, en base a otros datos de dicha área del marco de pila.

5. Un método de acuerdo con una cualquiera de las reivindicaciones 2 a 4, **caracterizado porque** cada marco de pila (10) está dividido en dos áreas, correspondientes respectivamente a un área variable local (11) del método convocado y un área de pila de operandos (13) de dicho método.

6. Un método de acuerdo con una cualquiera de las reivindicaciones precedentes, **caracterizado porque** es implementado en forma de software.

7. Un método de acuerdo con una cualquiera de las reivindicaciones 1 a 5, **caracterizado porque** es implementado en forma de hardware.

8. Un aparato digital que comprende una memoria no volátil, comprendiendo dicha memoria una máquina virtual para la ejecución de una aplicación compilada en un código intermedio, **caracterizado porque** dicha máquina virtual está adaptada para implementar el método de acuerdo con una cualquiera de las reivindicaciones precedentes.

9. Un aparato de acuerdo con la reivindicación 8, **caracterizado porque** dicho dispositivo es una tarjeta inteligente.

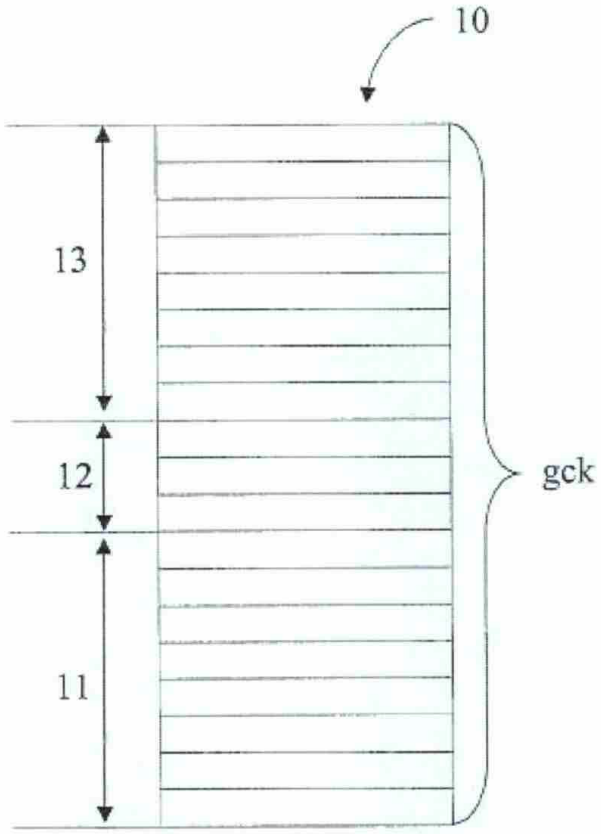


Fig.1

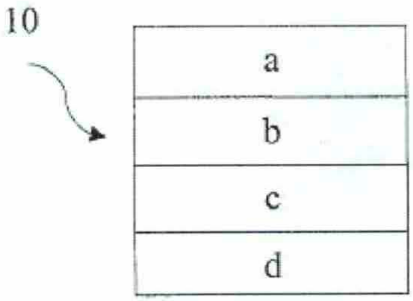


Fig.2a

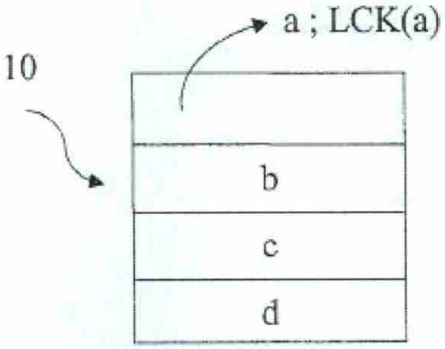


Fig.2b

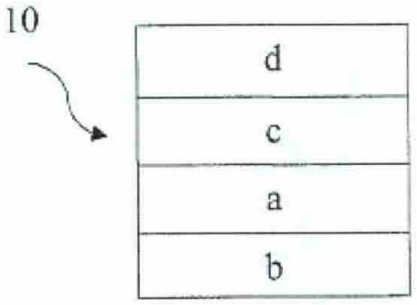


Fig.3a

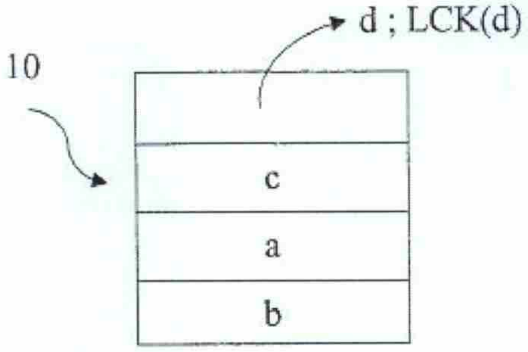


Fig.3b

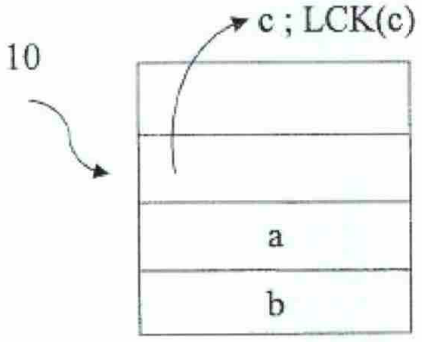


Fig.3c

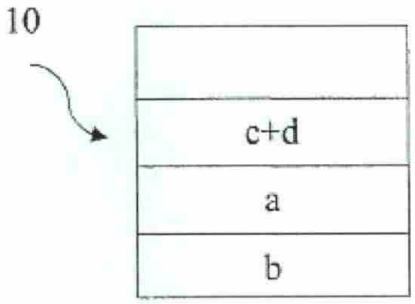


Fig.3d

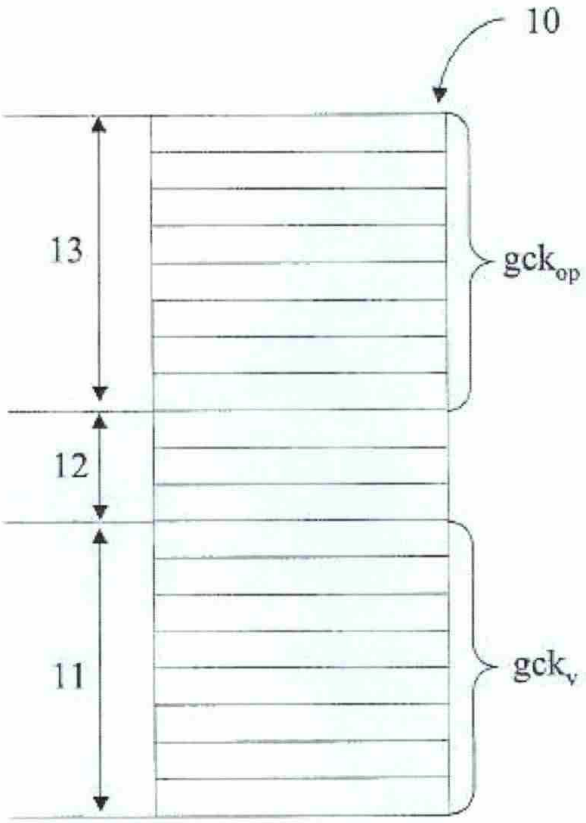


Fig.4

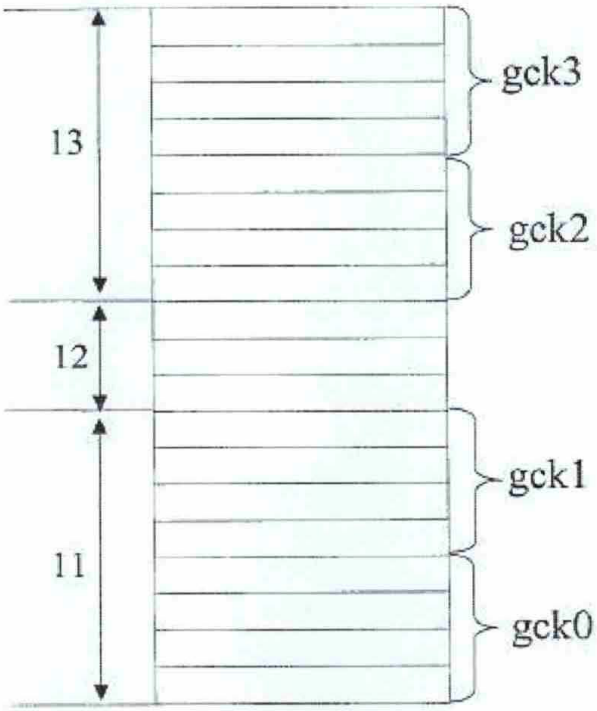


Fig.5

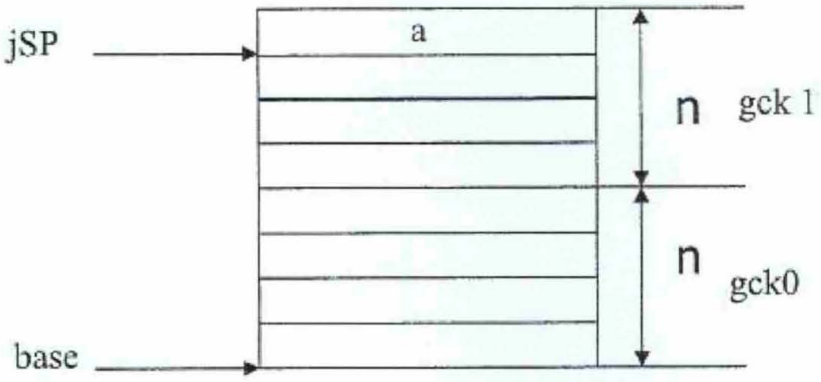

jSP
a
n
gck 1
n
gck0
base

Fig.6

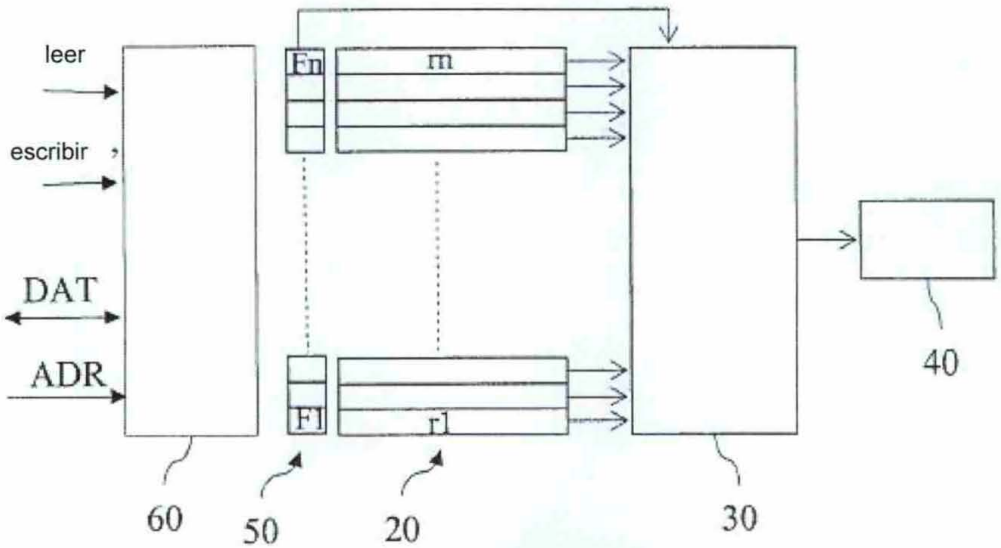

leer
escribir
DAT
ADR
En
m
Fl
rl
60
50
20
30
40

Fig.7