(19) Deutsches Patent- und Markenamt

(10) **DE 10 2012 015 899 A1** 2014.02.13

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2012 015 899.8**
(22) Anmeldetag: **10.08.2012**
(43) Offenlegungstag: **13.02.2014**

(51) Int Cl.: ***G06F 9/44*** (2006.01)

(71) Anmelder:
Giesecke & Devrient GmbH, 81677, München, DE

(72) Erfinder:
Baldischweiler, Michael, 81825, München, DE;
Stocker, Thomas, 81739, München, DE

(56) Ermittelter Stand der Technik:

FR	**2 818 846**	**A1**
US	**6 668 325**	**B1**
US	**2009 / 0 010 424**	**A1**
US	**2011 / 0 246 789**	**A1**
EP	**2 009 572**	**A1**

KEERSEBILCK, Philip et al.: SMART CARD (IN-) SECURITY.In: 8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMSSuceava, Romania, 25 – 27. Mai 2006, S. 249-254URL: http://www.dasconference.ro/papers/2006/C18.pdf [abgerufen am 09.04.2013]

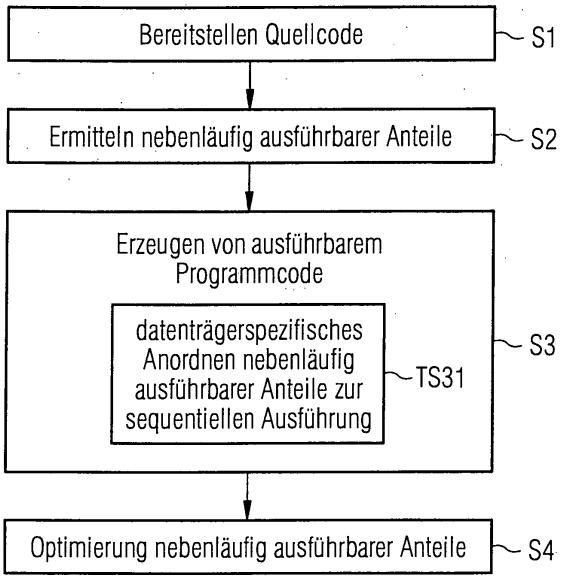
Witteman, M: Advances in Smartcard Security. In: Information Security Bulletin, Juli 2002, S. 11-22.URL: http://profs.info.uaic.ro/~fltiplea/IS/Witt2002.pdf[abgerufen am 09.04.2013]

Rechercheantrag gemäß § 43 Abs. 1 Satz 1 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Verfahren zum Erzeugen von ausführbarem Programmcode**

(57) Zusammenfassung: In einem Verfahren zum Erzeugen eines auf einem portablen Datenträger ausführbaren Programmcodes wird in einem ersten Schritt (S1) ein dem Programmcode (52) zugrunde liegender Quellcode bereitgestellt. Dann werden nebenläufig ausführbare Anteile (B, C, D) der durch den Quellcode definierten Operationen ermittelt (S2). In einem weiteren Schritt (S3) wird ausführbarer Programmcode aus dem Quellcode erzeugt. Dabei werden solche ausführbaren Programmcodeanteile, die zuvor ermittelten, nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes (52) Sicherheitsmodulspezifisch, vorzugsweise randomisiert, angeordnet (TS31).



Beschreibung

[0001] Die vorliegende Erfindung betrifft ein Verfahren zum Erzeugen von auf einem Sicherheitsmodul ausführbarem Programmcode sowie eine Menge entsprechender Sicherheitsmodule zum Ausführen des Programmcodes.

[0002] Im Zusammenhang mit der vorliegenden Erfindung ist der Begriff des ausführbaren Programmcodes derart breit auszulegen, dass jedwede Form von Programmcode darunterfällt, der von Prozessoren, Interpretern, virtuellen Maschinen oder dergleichen ausgeführt bzw. interpretiert werden kann. Insofern ist darunter Programmcode zu verstehen, welcher gemäß einem so genannten Zwischencode codiert ist. Als Zwischencode wird vorliegend ein Code in einer Sprache verstanden, welche konzeptionell zwischen einem Code eines Quelltextes einer Hochsprache, beispielsweise „Java", „C" oder dergleichen, auf der einen Seite und einer maschinennahen Zielsprache, insbesondere einem Maschinencode, auf der anderen Seite angeordnet ist.

[0003] Zwischencode kann von einem Compiler beispielsweise im Rahmen einer Übersetzung (Compilierung) eines Quelltextes hin zu Maschinencode als Zwischenergebnis erzeugt werden. Bekanntes Beispiel eines solchen Zwischencodes ist der so genannte „3-Adress-Code". Zum Ausführen dieses Codes sind verschiedene Interpreter bekannt. Weitere bekannte Beispiele für ausführbaren Zwischencode, welcher von einem entsprechenden Compiler vorwiegend zum plattformunabhängigen Ausführen durch einen „virtuellen" Prozessor, beispielsweise eine Virtuellen Maschine, erzeugt wird, sind so genannte Bytecodes, wie z. B. Java- oder Java Card™-Bytecode.

[0004] Anderseits kann unter „ausführbarem Programmcode" vorliegend auch Programmcode verstanden werden, welcher gemäß einer maschinennahen Zielsprache, insbesondere in Maschinencode, codiert ist. Programmcode dieser Art kann dann direkt von einem Hardware-Prozessor des Sicherheitsmoduls ausgeführt werden.

[0005] In neuerer Zeit stellen Angriffe auf Sicherheitsmodule, insbesondere Chipkarten, ein erhöhtes Sicherheitsrisiko sowohl für den individuellen Nutzer des Sicherheitsmoduls als auch für einen Hersteller der Sicherheitsmodule dar. Die Art der Angriffe ist mannigfaltig. Sie reicht beispielsweise von bloßem Abhören einer Datenkommunikationsverbindung zwischen einem solchen Sicherheitsmodul und einem Endgerät über das Auswerten des Ressourcenverbrauchs des Sicherheitsmoduls beim Durchführen einer Berechnung, beispielsweise des Stromverbrauchs oder der Rechenzeit, bis hin zu technisch sehr aufwendigen Angriffen. Dabei werden beispielsweise Berechnungen des Prozessors des Sicherheitsmoduls gezielt dadurch gestört, dass mittels externen Einwirkens auf das Sicherheitsmodul, beispielsweise mittels gezielter Lichtblitze auf einzelne Speicherzellen eines Speichers des Sicherheitsmoduls, Eingabedaten oder Zwischenergebnisse einer Berechnung manipuliert werden, um anhand der dadurch erhaltenen, manipulierten Ergebnisse auf in dem Sicherheitsmodul gespeicherte sensible Daten, beispielsweise Verschlüsselungsschlüssel, zurückzuschließen.

[0006] Aus Sicht des individuellen Nutzers können bei einem solchen Angriff persönliche oder sonstige sensible Daten, beispielsweise geheime Schlüssel, ausgespäht werden. Für einen Hersteller der Sicherheitsmodule kann bereits ein erfolgreicher Angriff auf ein Sicherheitsmodul einer Serie baugleicher Sicherheitsmodule, neben dem Imageverlust, einen erheblichen Schaden bedeuten, da dann möglicherweise die ganze Serie von Sicherheitsmodulen als kompromittiert angesehen und daher zurückgerufen oder ersetzt werden muss.

[0007] Dies kann für einen Hersteller auch in dem Fall problematisch sein, in dem Sicherheitsmodul aus an sich verschiedenen Anwendungsbereichen, beispielsweise aus den Bereichen Telekommunikation, Banking, PayTV, etc., auf einer identischen Produktlinienarchitektur beruhen. Solche Sicherheitsmodule haben dann hinsichtlich der Ausführung von jeweils identisch darin vorliegendem Programmcode, beispielsweise von Teilen des Betriebssystems, dasselbe physikalische Verhalten, so dass ein erfolgreicher Angriff auf ein Sicherheitsmodul aus einem Anwendungsbereich, beispielsweise eine Kreditkarte, negative Auswirkungen auf die Sicherheit eines Sicherheitsmoduls aus einem ganz anderen Bereich, beispielsweise eine SIM-Karte, haben kann.

[0008] Aufgabe der vorliegenden Erfindung ist es, Sicherheitsmodule mit darauf ausführbarem Programmcode gegen externe Angriffe zu schützen. Insbesondere sollen die Auswirkungen, welche ein erfolgreicher Angriff auf einen Sicherheitsmodul für andere, baugleiche Sicherheitsmodule hat, begrenzt werden.

[0009] Diese Aufgabe wird durch ein Verfahren und eine Menge von Sicherheitsmodulen mit den Merkmalen der unabhängigen Ansprüche gelöst. Vorteilhafte Ausgestaltungen und Weiterbildungen sind in den abhängigen Ansprüchen angegeben.

[0010] Die vorliegende Erfindung basiert auf der Grundidee, für ein individuelles Sicherheitsmodul oder für Gruppen von Sicherheitsmodulen jeweils einen sicherheitsmodulspezifischen, d. h. sicherheitsmodul- oder gruppenindividuellen ausführbaren Programmcode in der folgenden Weise zu erzeugen:

In einem ersten Schritt wird ein dem Programmcode zugrunde liegender Quellcode, vorzugsweise in einer Hochsprache, wie z. B. „C", „Java" oder dergleichen, bereitgestellt.

[0011] Dann werden nebenläufig ausführbare Anteile der durch den Quellcode definierten Operationen ermittelt. Diese Anteile können dabei bereits auf der Ebene des Quellcodes ermittelt werden. Es ist aber auch möglich, dass dieser Schritt erst auf der Ebene eines Zwischencodes zwischen dem Quellcode und dem ausführbaren Programmcode oder erst auf der Ebene des ausführbaren Programmcodes stattfindet. Schließlich können nebenläufig ausführbare Anteile, mit Bezug auf denselben zu erzeugenden Programmcode, auch auf verschiedenen dieser Ebenen ermittelt werden. D. h. das Ermitteln der Anteile kann zu verschiedenen Zeitpunkten und in verschiedenen Phasen einer Übersetzung des Quellcodes zu ausführbarem Programmcode durchgeführt werden.

[0012] In einem weiteren Schritt wird ausführbarer Programmcode aus dem Quellcode erzeugt.

[0013] Erfindungsgemäß werden im Schritt des Erzeugen des ausführbaren Programmcodes ausführbare Programmcodeanteile, die zuvor ermittelten, nebenläufig ausführbaren Anteilen entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes sicherheitsmodulspezifisch angeordnet.

[0014] Eine solche Anordnung der ausführbaren Programmcodeanteile bestimmt die Reihenfolge, in der die Anteile bei einer sequentiellen Ausführung des derart erzeugten Programmcodes ausgeführt werden, mithin das Laufzeitverhalten der durch den ausführbaren Programmcode bereitgestellten Funktionalität. Es versteht sich, dass diese Anordnung dabei derart erfolgt, dass die Gesamtfunktionalität, welche durch den erzeugten, ausführbaren Programmcode bereitgestellt wird, nicht beeinträchtig wird und für jede der möglichen, erzeugbaren, sicherheitsmodulspezifischen Anordnungen identisch ist.

[0015] Da jeder erzeugte Programmcode sicherheitsmodulspezifisch ist, d. h. gruppenspezifisch oder sogar sicherheitsmodulindividuell, zeigt ein Sicherheitsmodul einer ersten Gruppe beim Ausführen des für diese erste Gruppe erzeugten, ausführbaren Programmcodes ein anderes Laufzeitverhalten als ein Sicherheitsmodul einer zweiten Gruppe beim Ausführen des für diese zweite Gruppe erzeugten – funktional identischen, aber bezüglich des beobachtbaren Laufzeitverhaltens abweichenden – ausführbaren Programmcodes.

[0016] Auf diese Weise werden die jeweiligen Sicherheitsmodule gegen externe Angriffe geschützt. Insbesondere kann ein Angreifer aus einem erfolgreichen Angriff auf ein Sicherheitsmodul der ersten Gruppe in der Regel keine Informationen gewinnen, welche für Angriffe auf Sicherheitsmodule der zweiten Gruppe nützlich sein könnten.

[0017] Als „nebenläufig ausführbar" werden im Zusammenhang mit der vorliegenden Erfindung solche Anteile der durch den Quellcode definierten Operationen verstanden, die paarweise keine kausale Beziehung zueinander aufweisen. Eine solche kausale Beziehung könnte insbesondere darin bestehen, dass eine Berechnung, welche durch Programmcodeanteile entsprechend einem ersten Anteil durchgeführt wird, von Ergebnissen einer Berechnung, welche durch Programmcodeanteile entsprechend einem zweiten Anteil erfolgt, direkt oder indirekt abhängt oder beeinflusst wird. Zwei Anteile, beispielsweise zwei Transaktionen, Prozesse oder Threads, sind somit genau dann nebenläufig ausführbar oder parallelisierbar, wenn eine zeitlich parallele Ausführung, eine verzahnte Ausführung oder eine sequentielle Ausführung in geänderter Reihenfolge zu demselben Resultat führt wie ein sequentielles Ausführen gemäß einer vorgegebenen Reihenfolge.

[0018] Dabei kann es Anteile geben, die in gewissem Sinne „global" nebenläufig ausführbar sind, d. h. Anteile, die zu keinem anderen Anteil der durch den Quellcode definierten Operationen kausal in Beziehung stehen. Im Allgemeinen aber wird der Begriff „Nebenläufigkeit" als relativer Begriff in dem Sinne verstanden werden müssen, dass eine vorgegebene Menge von Anteilen, paarweise und relativ zueinander, nebenläufig, d. h. beispielweise zeitlich parallel oder in abgeänderter Reihenfolge, ausgeführt werden können. Dies sagt dann allerdings nichts darüber aus, ob, und wenn ja, welche kausale Beziehung zwischen einem oder mehreren dieser Anteile zu anderen, nicht zu der entsprechenden, vorgegebenen Menge von Anteilen gehörenden Anteilen bestehen. Schließlich können nebenläufig ausführbare Anteile auch in verschachtelter Weise auftreten, beispielweise im Falle einer Menge nebenläufig ausführbarer Anteile, die einen oder mehrere Anteile umfassen, welche ihrerseits aus nebenläufig ausführbaren Unteranteilen bestehen.

[0019] Die allgemeine Formulierung „Ermitteln von nebenläufig ausführbaren Anteilen" umfasst somit jede dieser Arten von global und/oder relativ zu anderen Anteilen nebenläufig ausführbaren Anteilen. In gleicher Weise gilt, dass im Schritt des sicherheitsmodulspezifischen Anordnen von ausführbaren Programmcodeanteilen, die den ermittelten, nebenläufig ausführbaren Anteilen entsprechen, die Art und der Umfang der nebenläufigen Ausführbarkeit beachtet werden. D. h. eine Neuanordnung eines ausführbaren Programmcodeanteils im erzeugten Programmcode zur nachfolgenden sequentiellen Ausführbarkeit erfolgt lediglich stets relativ zu den entsprechenden,

mit Bezug auf diesen ausführbaren Programmcodeanteil nebenläufig ausführbaren Programmcodeanteilen.

[0020] Gemäß einer bevorzugten Ausführungsform des vorstehend beschriebenen Verfahrens werden die ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes randomisiert angeordnet. Dies ermöglicht auf sehr einfache und für einen Angreifer nicht vorhersehbare Weise das Erzeugen von sicherheitsmodulspezifischem, ausführbarem Programmcode. Es ist allerdings auch möglich, dass das Anordnen deterministisch erfolgt oder dass eine Vorschrift zum sicherheitsmodulspezifischen Anordnen sowohl deterministische als auch zufällige Elemente umfasst.

[0021] Wie bereits erwähnt, hat ein sicherheitsmodulspezifisches Anordnen derjenigen ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen entsprechen, die Auswirkung, dass sich bei einer sequentiellen Ausführung des Programmcodes auf dem Sicherheitsmodul ein sicherheitsmodulspezifisches Laufzeitverhalten zeigt. Beobachtungen und Analysen eines Angreifers hinsichtlich des Ressourcenverbrauchs eines Sicherheitsmoduls, welcher sich beim Ausführen einer durch den erfindungsgemäß erzeugten, ausführbaren Programmcode bereitgestellten Funktionalität beobachten lässt, beispielsweise hinsichtlich der Laufzeit oder des Stromverbrauchs, sind damit ebenfalls sicherheitsmodulspezifisch und nicht auf andere Sicherheitsmodule oder andere Gruppen von Sicherheitsmodulen, welche dieselbe Funktionalität umfassen, übertragbar.

[0022] Ein weiterer vorteilhafter technischer Effekt der Erfindung, welcher die Sicherheit eines Sicherheitsmoduls einer ersten Gruppe auch bei einem erfolgreichen Angriff auf ein Sicherheitsmodul einer anderen Gruppe gewährleistet, ist die Tatsache, dass eine beschriebene, sicherheitsmodulspezifische Anordnung von ausführbaren Programmcodeanteilen, welche nebenläufig ausführbaren Anteilen entsprechen, beim Speichern des Programmcodes in einem nichtflüchtigen Speicher des Sicherheitsmoduls in der Regel ein sicherheitsmodulspezifisches Speicherabbild ergibt. Damit kann aus einem erfolgreichen Angriff auf eine Funktionalität eines Sicherheitsmoduls, welche durch gezielte Manipulation einzelner Speicherzellen erfolgt ist, kein Vorteil für einen entsprechenden Angriff auf ein weiteres Sicherheitsmodul mit der gleichen Funktionalität gezogen werden, da dort die entsprechenden Speicherzellen in der Regel, als Folge der erfindungsgemäßen sicherheitsmodulspezifischen Anordnung der Anteile des die Funktionalität bereitstellenden ausführbaren Programmcodes, abweichend belegt sind.

[0023] Im Rahmen des Schritts des Ermitteln von nebenläufig ausführbaren Anteilen kann eine Anzahl von gleichzeitig nebenläufig ausführbaren Anteilen vorgegeben werden. Diese Anzahl von gleichzeitig nebenläufig oder parallel ausführbaren Anteilen entspräche, wenn Programmcode zum Ausführen auf einer Mehrkern- oder Mehrprozessorarchitektur erzeugt werden sollte, beispielsweise der Anzahl der dort bereitstehenden Kerne oder Prozessoren. Je größer diese Anzahl gewählt oder vorgegeben wird, desto großer ist die Anzahl von möglichen Anordnungen entsprechender Anteile für eine sequentielle Ausführung des zu erzeugenden Programmcodes.

[0024] Diejenigen ausführbaren Programmcodeanteile, die ermittelten, nebenläufig ausführbaren Anteilen entsprechen, können beim Erzeugen des ausführbaren Programmcodes in einer Tabelle angeordnet werden. Dabei kann die Anordnung innerhalb der Tabelle direkt sicherheitsmodulspezifisch, bevor zugt randomisiert, erfolgen. Es ist aber auch möglich, zuerst eine Tabelle zu erstellen, welche die entsprechenden Programmcodeanteile in einer vorgegebenen Reihenfolge umfasst, beispielsweise in der Reihenfolge, in der die Anteile ermittelt worden sind. In einem nachfolgenden Schritt kann dann eine sicherheitsmodulspezifische Anordnung dieser Anteile, beispielsweise durch Permutation der Tabelleneinträge, erfolgen. Die Reihenfolge, in der die Programmcodeanteile am Ende eines der beschriebenen Prozesse in der Tabelle angeordnet sind, bestimmt die Ausführungsreihenfolge der mittels der Programmcodeanteile codierten implementierten Teilfunktionalitäten bei sequentieller Ausführung des erzeugten Programmcodes.

[0025] Grundsätzlich können nebenläufig ausführbare Anteile der durch den bereitgestellten Quellcode definierten Operationen in allgemeiner Weise, auch aus einem bereits für eine sequentielle Ausführung angelegten Programmcode, ermittelt werden. Einige bekannte Programmstrukturen eignen sich allerdings besonders zur Parallelisierung, wie beispielsweise Schleifen. D. h. als nebenläufig ausführbare Anteile der durch den Quellcode definierten Operationen können bevorzugt eine Schleifenoperation definierende Anteile ermittelt werden. Eine Schleife beispielsweise der Form (for i = 0; i < 1000; i++){a [i] := i} kann in bis zu 1000 Anteile aufgeteilt werden, jeder Anteil entsprechend zumindest einem Schleifendurchlauf, welche jeweils nebenläufig ausgeführt werden können. In gleicher Weise können als nebenläufig ausführbare Anteile eine geeignete Rekursion definierende Anteile ermittelt werden. Dies gilt insbesondere dann, wenn in einem Rekursionsaufruf dieselbe Funktion mehr als einmal, unabhängig voneinander und mit verschiedenen Parametern, aufgerufen wird, wie dies beispielsweise bei einer bekannten, rekursiven Implementation des Quick-Sort-Algorithmus der Fall ist.

[0026] Gemäß einer weiteren bevorzugten Ausführungsform kann zumindest einer der ausführbaren Programmcodeanteile, der einem nebenläufig ausführbaren Anteil entspricht, im Schritt des Erzeugen des ausführbaren Programm codes hinsichtlich zumindest eines vorgegebenen Parameters optimiert werden. Bevorzugte Parameter sind beispielsweise Laufzeit und/ oder Speicherbedarf. Diese Optimierung kann vor oder nach der sicherheitsmodulspezifischen Anordnung mit Bezug auf die anderen ausführbaren Programmcodeanteile erfolgen, welche ebenfalls nebenläufig ausführbaren Anteilen entsprechen. Die Optimierung kann weiterhin in einer oder mehreren verschiedenen Phasen eines Übersetzungsprozesses vom Quellcode zum ausführbaren Programmcode erfolgen, also beispielsweise auch bereits auf Ebene eines Zwischencodes.

[0027] Auf diese Weise kann der erzeugte ausführbare Programmcode, im Vergleich zu einem entsprechenden Programmcode ohne Optimierung, ebenfalls sicherheitsmodulspezifisch ausgebildet werden. Eine Optimierung bezüglich Laufzeit, auch wenn diese nur einzelne Teilfunktionalitäten betrifft, verändert das Laufzeitverhalten der durch den erzeugten, ausführbaren Programmcode bereitgestellten Gesamtfunktionalität. Eine – auch nur teilweise – Optimierung bezüglich des Speicherbedarfs beeinflusst das gesamte Speicherabbild. Eine solche Optimierung kann überdies die Behandlung lokaler Variablen im Bereich der optimierten, ausführbaren Programmcodeanteile beeinflussen, und somit während der Laufzeit beispielsweise auch die Belegung eines Stapelspeichers, verschiedener Prozessorregister oder anderer Speicherbereiche, beispielsweise im Arbeitsspeicher des Prozessors.

[0028] Besonders bevorzugt werden verschiedene der ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen entsprechen, jeweils hinsichtlich verschiedener Parameter optimiert. Auf diese Weise ergeben sich vielfältige, weitere Möglichkeiten des sicherheitsmodulspezifischen Erzeugen des ausführbaren Programmcodes.

[0029] Gemäß einer weiteren bevorzugten Ausführungsform des erfindungsgemäßen Verfahrens wird der Schritt des Ermitteln von nebenläufig ausführbaren Anteilen der durch den Quellcode definierten Operationen von einer Einrichtung durchgeführt oder durch eine solche Einrichtung zumindest unterstützt, welche zum Erzeugen von auf einer Mehrkern- und/ oder Mehrprozessorarchitektur parallel ausführbarem Programmcode eingerichtet ist. Eine solche Einrichtung kann auf verschiedenen Ebenen eingreifen oder wirken, beispielsweise in Form von in den Quellcode einfügbaren Compilerdirektiven oder, bei unverändertem Quellcode, durch Funktionalitäten des Compilers selbst.

[0030] D. h. eine solche bereits vorliegende, bekannte Einrichtung, beispielsweise ein Compiler zum Erzeugen von parallel ausführbarem Programmcode, wird erfindungsgemäß „zweckentfremdet", indem durch diese Einrichtung erzeugter, parallel auf einer Mehrprozessorarchitektur ausführbarer Programmcode als Zwischenergebnis verwendet wird. Der aufwendige und in der Regel komplexe Schritt des Ermitteln von nebenläufig ausführbaren Anteilen wird vollständig von dieser Einrichtung übernommen und, in der Regel zusätzlich konfigurierbar, durchgeführt.

[0031] Zum erfindungsgemäßen Erzeugen des sequenziell ausführbaren, sicherheitsmodulspezifischen Programmcodes ist es abschließend lediglich erforderlich, die von der Einrichtung für eine jeweils parallele Ausführung vorgesehenen Anteile, wie beispielsweise parallel ausführbare Threads oder dergleichen, sicherheitsmodulspezifisch, vorzugsweise randomisiert, für eine sequentielle Ausführung auf einem Datenträger mit nur einem Prozessor anzuordnen. Dabei wird ein vermeintlicher Nachteil der parallelen Programmierung, nämlich das notwendig daraus resultierende, nicht genau vorhersehbare Laufzeitverhalten einer entsprechenden, parallelisierten Anwendung, erfindungsgemäß in einen Vorteil umgemünzt, indem nämlich die Nichtvorhersehbarkeit des Laufzeitverhaltens, welche erfindungsgemäß ein Sicherheitsmerkmal des Sicherheitsmoduls ist, durch eine sicherheitsmodulspezifische Anordnung parallelisierbarer Anteile zur sequentiellen Ausführung noch verstärkt wird.

[0032] Jedes Sicherheitsmodul einer erfindungsgemäßen Menge von Sicherheitsmodulen umfasst einen Speicher und einen Prozessor zum Ausführen von in dem Speicher gespeichertem, ausführbarem Programmcode. Die erfindungsgemäße Menge von Sicherheitsmodulen umfasst zumindest zwei Gruppen von Sicherheitsmodulen, wobei jede dieser Gruppen zumindest ein Sicherheitsmodul umfasst. Erfindungsgemäß ist der jeweils in einem Speicher eines Sicherheitsmoduls der Menge von Sicherheitsmodulen gespeicherte, ausführbare Programmcode hinsichtlich dessen Funktionalität für jedes Sicherheitsmodul der Menge von Sicherheitsmodulen identisch, unterscheidet sich jedoch hinsichtlich dessen Laufzeitverhalten für jeweils zwei Sicherheitsmodule aus unterschiedlichen Gruppen der Gruppen von Sicherheitsmodulen.

[0033] Wie vorstehend bereits beschrieben, gewährleistet ein sicherheitsmodulspezifisches, d. h. gruppen- oder sicherheitsmodulindividuelles Laufzeitverhalten einer durch den ausführbaren Programmcode bereitgestellten Funktionalität die Sicherheit eines Sicherheitsmoduls einer Gruppe auch im Hinblick auf einen erfolgreichen Angriff auf ein Sicherheitsmodul einer anderen Gruppe.

[0034] Wie mit Bezug auf das erfindungsgemäße Verfahren beschrieben, beruht das unterschiedliche Laufzeitverhalten des Programmcodes für zwei Sicherheitsmodule aus unterschiedlichen Gruppen insbesondere darauf, dass sich der Programmcode eines Sicherheitsmoduls einer ersten Gruppe von dem Programmcode eines Sicherheitsmoduls einer zweiten Gruppe darin unterscheidet, dass beim Erzeugen des ausführbaren Programmcodes für die Sicherheitsmodule jeweils ausführbare Programmcodeanteile, die nebenläufig ausführbaren Anteilen entsprechen, für eine sequenzielle Ausführbarkeit des ausführbaren Programmcodes durch den Prozessor des jeweiligen Sicherheitsmoduls gruppenspezifisch angeordnet worden sind. In der Regel wird sich dabei in einem nichtflüchtigen Speicher des Sicherheitsmoduls weiterhin ein sicherheitsmodulspezifisches Speicherabbild ergeben.

[0035] Das vorliegende System zur Erstellung von ausführbarem Programmcode umfasst ein speziell angepasstes Werkzeug zum Erzeugen des sequentiell ausführbaren Programmcodes für einen Quellcode und ein Werkzeug zur Ermittlung von nebenläufig ausführbaren Anteilen. Das Erzeugungswerkzeug ist eingerichtet die von dem Werkzeug zur Ermittlung ermittelten nebenläufig ausführbaren Programmcodeanteile für eine sequenzielle Ausführung auf dem Sicherheitsmodul bereitzustellen. Vorzugsweise kann ein Multiprozessor-Werkzeug zur Ermittlung verwendet werden. Das Erzeugungswerkzeug ist dagegen ein Monoprozessor-Werkzeug. Das Ermittlungswerkzeug kann die ermittelten nebenläufig ausführbaren Programmcodeanteile für die nebenläufige Ausführung aufbereiten. Die Aufbereitung ergänzt insbesondere die für eine nebenläufige Ausführung nötigen Operationen (Anweisungen), beispielsweise um jedem Prozessor bzw. Programmcodeanteil die (zumindest teilweise gleichen) erforderlichen Werte (Variablen, Zeiger ...) zur Verfügung zu stellen.

[0036] Ein Sicherheitsmodul ist vorzugsweise ein Hardware-Sicherheitsmodul. Das Sicherheitsmodul kann ein tragbarer Datenträger sein, wie beispielsweise eine Chipkarte, ein USB-Token, ein RFID-Token oder eine sichere Massenspeicherkarte. Das Sicherheitsmodul kann reversibel mit einem Endgerät verbunden werden oder fest in ein Endgerät eingebaut sein, wie beispielsweise ein eingebautetes SIM-Modul (eUICC), ein TPM-Modul oder NFC-Modul.

[0037] Die vorliegende Erfindung wird im Folgenden mit Bezug auf die beiliegenden Zeichnungen beispielhaft beschrieben. Darin zeigen:

[0038] **Fig.** 1 eine bevorzugte Ausführungsform eines Datenträgers aus einer Variante einer erfindungsgemäßen Menge von Datenträgern;

[0039] **Fig.** 2 Schritte einer bevorzugten Ausführungsform eines erfindungsgemäßen Verfahrens zum Erzeugen ausführbaren Programmcodes;

[0040] **Fig.** 3A bis **Fig.** 3E Ergebnisse einzelner Teilschritte von Verfahrensschritten aus **Fig.** 2 und

[0041] **Fig.** 4 die Menge von Datenträgern, unterteilt in einzelne Gruppen, mit dem Datenträger aus **Fig.** 1 als Element einer Gruppe.

[0042] Mit Bezug auf **Fig.** 1 umfasst ein Datenträger **10** als Sicherheitsmodul, der hier als Chipkarte dargestellt ist, Datenkommunikationsschnittstellen **20**, **20'**, einen Prozessor **30** sowie verschiedene Speicher **40**, **50** und **60**. Der Datenträger **10** kann auch in anderer Bauform vorliegen.

[0043] Als Datenkommunikationsschnittstellen **20**, **20'** umfasst der Datenträger **10** ein Kontaktfeld **20** zur kontaktbehafteten Datenkommunikation sowie eine Antennenspule **20'** zur kontaktlosen Datenkommunikation. Alternative Datenkommunikationsschnittstellen können vorgesehen sein. Es ist weiterhin möglich, dass der Datenträger **10** lediglich eine Art der Datenkommunikation unterstützt, also lediglich kontaktbehaftet oder kontaktlos.

[0044] Der nicht flüchtige, nicht wiederbeschreibbare ROM-Speicher **40** umfasst ein Betriebssystem (OS) **42** des Datenträgers **10**, welches den Datenträger **10** steuert. Zumindest Teile des Betriebssystems **42** können auch in dem nicht flüchtigen, wiederbeschreibbaren Speicher **50** gespeichert sein. Dieser kann beispielsweise als FLASH-Speicher vorliegen.

[0045] Der Speicher **50** umfasst datenträgerspezifischen, ausführbaren Programmcode **52**, welcher, wie nachfolgend mit Bezug auf die **Fig.** 2 bis **Fig.** 3D beschrieben, erzeugt worden ist.

[0046] Der flüchtige, wiederbeschreibbare RAM-Speicher **60** dient dem Datenträger **10** als Arbeitsspeicher.

[0047] Wie in **Fig.** 4 gezeigt, liegen weitere Datenträger **10''**, **10'''** einer Menge **100** von Datenträgern, unterteilt in verschiedene Gruppen **101**, **102**, **103**, vor. Datenträger innerhalb einer Gruppe sind hinsichtlich der beschriebenen Komponenten jeweils identisch. Datenträger **10**, **10''**, **10'''** unterschiedlicher Gruppen **101**, **102**, **103** unterscheiden sich darin, dass die Funktionalität, welche durch den ausführbaren Programmcode **52**, **52''**, **52'''** (vgl. **Fig.** 3C, **Fig.** 3E) bereitgestellt wird, jeweils ein gruppenspezifisches Laufzeitverhalten aufweist und gegebenenfalls zu einem gruppenspezifischen Speicherabbild in einem nichtflüchtigen Speicher des jeweiligen Datenträgers führt. Die durch den Programmcode **52**, **52'**, **52''**, **52'''** bereitgestellte Funktionalität ist allerdings für alle Da-

tenträger **10**, **10''**, **10'''** der Menge **100** von Datenträgern, unabhängig von der Gruppenzugehörigkeit **101**, **102**, **103**, identisch.

[0048] Mit Bezug auf **Fig.** 2 werden im Folgenden einzelne Schritte eines Verfahrens zum Erzeugen eines datenträgerspezifischen, ausführbaren Programmcodes **52** für den Datenträger **10** beschrieben.

[0049] In Schritt S1 wird ein Quellcode bereitgestellt. Dieser Quellcode definiert diejenigen Operationen, welche durchgeführt werden müssen, um die gewünschte Funktionalität bereitzustellen. Der Quellcode liegt in der Regel in einer Hochsprache, beispielsweise in „C", „Java" oder dergleichen vor.

[0050] In Schritt S2 werden nebenläufig ausführbare Anteile der durch den Quellcode definierten Operationen ermittelt. Ausführbarer Programmcode **52** wird in Schritt S3 erzeugt, wobei in Teilschritt TS31 nebenläufig ausführbare Anteile, wie nachfolgend beschrieben, datenträgerspezifisch, angeordnet werden. Dieser Teilschritt TS31 kann vor dem Erzeugen derjenigen ausführbaren Programmcodeanteile erfolgen, welche den als nebenläufig ausführbar ermittelten Anteilen entsprechen, oder aber danach. In einem optionalen vierten Schritt S4 können einzelne als nebenläufig ausführbar ermittelten Anteile und/oder die dazugehörigen, erzeugten, ausführbaren Programmcodeanteile nach verschiedenen Kriterien optimiert werden. Auch die Optimierung kann bereits während, d. h. im Rahmen des Übersetzungsprozesses, also beim Erzeugen des ausführbaren Programmcodes, erfolgen, bevor beispielsweise abschließend ausführbarer Maschinencode erzeugt wird.

[0051] Eine mögliche Vorgehensweise zum Ermitteln nebenläufig ausführbarer Anteile sowie zur Anordnung der entsprechenden Programmcodeanteile ist in den **Fig.** 3A bis **Fig.** 3C skizziert.

[0052] Sofern keine Missverständnisse zu erwarten sind, wird im Folgenden nur noch von „nebenläufig ausführbaren Anteilen" gesprochen, ohne genau zu unterscheiden, auf welcher Ebene diese gerade betrachtet werden. Dies kann beispielsweise die Ebene des Quellcodes, die Ebene eines Zwischencodes oder die Ebene eines Maschinencodes betreffen, je nachdem in welcher Phase eines nachfolgend grob skizzierten Übersetzungsprozesses diese Anteile bestimmt werden. D. h. eine Quellcodesequenz, welche beispielsweise eine Schleifenoperation codiert und in nebenläufig ausführbare Anteile zerlegt werden kann, wird in diesem Sinne mit der entsprechend in den Zwischencode übersetzten Sequenz bzw. mit dem Stück ausführbaren Maschinencode gleichgesetzt, welcher diese Schleifenoperation für den Prozessor codiert.

[0053] Das Ermitteln der nebenläufig ausführbaren Anteile erfolgt in der Regel im Rahmen einer Übersetzung oder Compilierung des Quellcodes in den ausführbaren Programmcode. Eine solche Übersetzung verläuft in bekannter Weise in verschiedenen Teilschritten. Diese können die Schritte der lexikalischen Analyse, der Syntaxanalyse, der semantischen Analyse, der Erzeugung von Zwischencode, der Codeoptimierung und der Codeerzeugung umfassen. Phasen, welche das Ermitteln der nebenläufig ausführbaren Anteile hauptsächlich betreffen können, sind beispielsweise die Syntaxanalyse, in der hierarchische Strukturen des Programms erkannt werden. Alternativ oder zusätzlich können nebenläufig ausführbare Anteile während der Erzeugung von Zwischencode ermittelt werden, bei der die Ergebnisse der vorhergehenden Schritte, welche zur Analyse des Quellcodes dienten, in eine „abstrakte Maschinensprache", einen geeigneten Zwischencode, wie z. B. den 3-Adress-Code, übersetzt werden, von dem aus dann nachfolgend der tatsächliche Maschinencode für die vorliegende Plattform erzeugt wird. Es ist aber auch möglich, dass nebenläufig ausführbare Anteile, zumindest grob, bereits auf der Ebene des Quellcodes, also vor der Übersetzung, oder erst auf der Ebene des Maschinencodes, d. h. am Ende der Übersetzung, ermittelt werden. Schließlich können verschiedene, nebenläufig ausführbare Anteile in verschiedenen der vorstehend beschriebenen Phasen oder in mehr als einer Phase bestimmt werden.

[0054] Ein erster Teilschritt während des Ermitteln von nebenläufig ausführbaren Anteilen, der in **Fig.** 3A illustriert ist, kann eine Zerlegung des Programmcodes, auf einer geeigneten Ebene des Programmcodes **51**, in Blöcke betreffen. Jeder dieser Blöcke A bis G umfasst jeweils eine Folge von Anweisungen, welche, wenn der Block erreicht wird, ausgeführt werden müssen, bevor der Block wieder verlassen wird. Es ist jedoch möglich, dass die Reihenfolge, in der einzelne der Blöcke ausgeführt werden können, variieren kann, ohne Einfluss auf das Endergebnis. In analoger Weise ist es möglich, dass Anweisungen innerhalb eines Blockes in verschiedener Reihenfolge ausgeführt werden können, ohne dass sich am Resultat der Berechnung etwas ändert. Diese beiden Fälle deuten auf nebenläufig ausführbare Anteile hin.

[0055] Wie mit Bezug auf **Fig.** 3B angedeutet, hat der bis dahin durchgeführte Übersetzungsprozess ergeben, dass die Blöcke A bis G nicht notwendig in der in **Fig.** 3A angedeuteten Reihenfolge ausgeführt werden müssen. Vielmehr zeigt sich, dass, nachdem die Anweisungen des Blocks A abgearbeitet worden sind, die Blöcke B, C und D parallel, verzahnt oder in beliebiger sequentieller Reihenfolge ausgeführt werden können, bevor das Programm mit Block E fortgesetzt wird. Mit anderen Worten, es besteht keine kausale Beziehung zwischen den Anteilen der Blöcke B, C und D. Keiner dieser Blöcke benötigt eine Einga-

be, welche in direkter oder indirekter Weise von einer Ausgabe eines der anderen Blöcke abhängen könnte. Daher können diese Blöcke als – relativ zueinander – nebenläufig ausführbar ermittelt werden. Das gleiche gilt für die Unteranteile F1, F2, ..., Fk des Anteils gemäß Block F.

[0056] Gemäß einer ersten Variante kann ausgehend von diesem Zwischenergebnis ausführbarer Programmcode für eine Mehrprozessorarchitektur, beispielsweise mit drei Prozessoren, erzeugt werden. Dabei würde beispielsweise für jeden der drei Blöcke B, C und D ausführbarer Programmcode erzeugt, welcher dann parallel auf jeweils einem der drei Prozessoren ausgeführt werden könnte. In analoger Weise könnte für die Unteranteile F1, F2, ..., Fk jeweils parallel ausführbarer Programmcode für einen der drei Prozessoren erzeugt werden.

[0057] Ausgehend von einem derart für eine Mehrprozessorarchitektur erzeugten, ausführbaren Programmcode kann sequentiell ausführbarer Programmcode für einen Datenträger **10** mit lediglich einem Prozessor erzeugt werden, wie dies in **Fig.** 3C illustriert ist. Dabei sind die Programmcodeanteile, welche den jeweils als nebenläufig ausführbar ermittelten Anteilen B, C und D bzw. F1, F2, ..., Fk, entsprechen, in zufälliger Reihenfolge für eine sequentielle Ausführung neu angeordnet worden. Aufgrund der Tatsache, dass diese Anteile parallel ausführbar sind, d. h. keine kausale Beziehung zueinander aufweisen, bleibt die Funktionalität des Gesamtprogramms identisch erhalten, egal in welcher Reihenfolge die Anordnung erfolgt. Dadurch, dass die Anordnung randomisiert erfolgt, ergibt sich insbesondere eine datenträgerspezifische Anordnung. Wird der entsprechende Programmcode **52** in einem nicht flüchtigen Speicher **50** des Datenträgers **10** gespeichert, ergibt sich ein datenträgerspezifisches Speicherabbild. Bei Ausführung des Programmcodes **52** ist ein datenträgerspezifisches Laufzeitverhalten erkennbar. Diese beiden technischen Effekte ergeben sich unmittelbar aus der Tatsache, dass die Programmcodeanteile, welche den nebenläufig ausführbaren Anteilen B, C, D bzw. F1, F2, ..., Fk entsprechen, datenträgerspezifisch angeordnet worden sind.

[0058] Gemäß einer zweiten Variante wird nicht erst, wie mit Bezug auf die erste Variante beschrieben, ausführbarer Programmcode für eine Mehrprozessorarchitektur erzeugt, welcher zum Erzeugen einer sequentiell ausführbaren Version erst wieder zerlegt und in beschriebener Weise neu angeordnet werden muss. Alternativ erfolgt hier die zufällige Anordnung der nebenläufig ausführbaren Anteile auf einer Ebene oberhalb des Maschinencodes, beispielsweise auf der Ebene des erwähnten Zwischencodes. Ausführbarer Programmcode **52** wird dann lediglich zur sequentiellen Ausführung erzeugt, wobei die den nebenläufig ausführbaren Anteilen B, C, D bzw. F1, F2, ..., Fk entsprechenden Programmcodeanteile erneut datenträgerspezifisch, wie in **Fig.** 3C gezeigt, angeordnet werden.

[0059] Die Resultate, d. h. der erzeugte Programmcode **52**, der ersten und der zweiten Variante entsprechen einander im Wesentlichen. Leichte Unterschiede können sich dadurch ergeben, dass beispielsweise Programmcode, welcher für eine echt parallele Ausführung auf einer Mehrprozessorarchitektur erzeugt worden ist, zusätzlichen Code umfasst, beispielsweise für eine mehrfache Initialisierung einzelner Variablen für jeden der parallel auszuführenden Zweige. Dieser Code ist in der direkt zur sequentiellen Ausführung erzeugten Version verzichtbar. Insofern ergibt sich ein jeweils unterschiedliches Speicherabbild. Hinsichtlich des Laufzeitverhaltens hingegen sind praktisch keine erkennbaren Unterschiede zu erwarten.

[0060] Grundsätzlich wird die Funktionalität zum Ermitteln nebenläufig ausführbarer Anteile durch einen entsprechend eingerichteten Compiler bereitgestellt, insbesondere durch einen Compiler, der eingerichtet ist, Programmcode zu erzeugen, welcher auf einer Mehrprozessorarchitektur echt parallel ausgeführt werden kann. Das beschriebene Verfahren zum Erzeugen des sequentiell ausführbaren, datenträgerspezifischen Programmcodes macht sich, wie beschrieben, einen solchen Compiler oder eine vergleichbare Einrichtung zunutze.

[0061] Es kann weiter vorgesehen sein, dass alternativ oder zusätzlich, auf der Ebene des Quellcodes bereits Anteile vorgegeben werden können, welche nebenläufig ausgeführt werden können. Dies kann beispielsweise mittels geeigneter Compiler-Direktiven erfolgen, die in den Programmcode geeignet eingefügt werden, wie diese z. B. im Zusammenhang mit der Programmierschnittstelle „OpenMP” bekannt sind. Mittels solcher Direktiven kann dem Compiler beispielsweise mitgeteilt werden, dass grundsätzlich als sequentiell ausführbar vorgesehene Anteile, so genannte „sections”, z. B. die vorstehend angegebenen Anteile B, C und D, nebenläufig ausgeführt werden können. Dabei obliegt es dem Programmierer, die Bedingungen der nebenläufigen Ausführbarkeit zu überprüfen.

[0062] Weiter können mittels geeigneter Direktiven solche Programmcodesequenzen, von denen bekannt ist, dass deren Anweisungen in einfacher Weise parallel ausgeführt werden können, dem Compiler angezeigt werden. Besonders geeignet sind Schleifenkonstruktionen, bei denen die einzelnen Schleifendurchläufe bezüglich Ein- und Ausgabedaten unabhängig voneinander sind. Diese Situation ist in den **Fig.** 3A und **Fig.** 3B mit dem Block F und den Unteranteilen F1, F2, ..., Fk angedeutet. Der Block F könnte beispielsweise die Schleife (for i = 0; i < k; i++){f[i] :=

2 × i + 1} umfassen. Mittels einer Compilerdirektive wird der Compiler angewiesen, die Schleife zu parallelisieren. Dieser erkennt, dass die einzelnen Schleifendurchläufe unabhängig voneinander sind und erzeugt entsprechend k nebenläufig ausführbare Unteranteile F1, F2, ..., Fk.

[0063] Es kann vorteilhaft sein, im Schritt des Ermitteln von nebenläufig ausführbaren Anteilen eine Anzahl von gleichzeitig nebenläufig ausführbaren Anteilen vorzugeben. Im vorhergehenden Beispiel hätte somit auch mit Bezug auf den Block F die Anzahl der Unteranteile beschränkt werden können.

[0064] Wie bereits erwähnt und mit Bezug auf **Fig.** 3D illustriert, können ein oder mehrere der als nebenläufig ausführbar ermittelten Anteile hinsichtlich eines oder mehrerer Parameter optimiert werden, beispielsweise die Anteile C und B, wobei sich die verschieden schraffiert hervorgehobenen optimierten Anteile C' und B'' ergeben. Bekannte Compiler verfügen dazu in der Regel über entsprechende Compile-Optionen, welche es beispielsweise erlauben, in mehreren Stufen hinsichtlich Ausführungsgeschwindigkeit oder Speicherbedarf optimierten Programmcode zu erzeugen. Dadurch, dass nicht der gesamte Programmcode einheitlich optimiert wird, sondern lediglich ausgewählte Anteile, kann eine weitere datenträgerspezifische Ausgestaltung des in dieser Weise erzeugten Programmcodes **52'** erreicht werden. Die Optimierung wirkt sich in der Regel sowohl auf das Laufzeitverhalten als auch auf die benötigten Speicherressourcen aus. Es können auch Optimierungsstufen gewählt werden, welche lediglich geeigneten, optimierten Zwischencode erzeugen. Ausführbarer Programmcode wird dann erst von einer Linker-Funktionalität erzeugt. Auf diese Weise kann einer Optimierung durchgeführt werden, welche nicht lediglich einzelne Module, sondern ein gesamtes Programmpaket betrifft.

[0065] Es versteht sich, dass die Optimierung alternativ oder zusätzlich auch auf solche Anteile A, E und G erstreckt werden kann, die nicht nebenläufig ausführbar sind.

[0066] In **Fig.** 3E ist exemplarisch ein für einen Datenträger **10''** (vgl. **Fig.** 4) erzeugter, ausführbarer, datenträgerspezifischer Programmcode **52''** gezeigt. Dieser unterscheidet sich von dem Programmcode **52'**, welcher für den Datenträger **10** erzeugt worden ist, einmal dahingehend, dass die den nebenläufig ausführbaren Anteilen B, C, D bzw. F1, F2, ..., Fk entsprechenden Programmcodeanteile für die sequentielle Ausführbarkeit gemäß einer abweichenden Reihenfolge angeordnet worden sind. Zum zweiten ist lediglich der Programmcodeanteil C optimiert worden, und zwar hinsichtlich eines anderen Parameters (beispielsweise Laufzeit anstelle von Speicherbedarf). Es ist unmittelbar einsichtig, dass das Laufzeitverhalten des Programmcodes **52''** des Datenträgers **10''** unterschiedlich ist von dem Laufzeitverhalten des Programmcodes **52'** des Datenträgers **10**.

Patentansprüche

1. Verfahren zur Erzeugung eines auf einem Sicherheitsmodul (**10**) ausführbaren Programmcodes (**52**), umfassend die Schritte:
– Bereitstellen (S1) eines Quellcodes;
– Ermitteln (S2) von nebenläufig ausführbaren Anteilen (B, C, D) der durch den Quellcode definierten Operationen;
– Erzeugen (S3) eines ausführbaren Programmcodes (**52**) aus dem Quellcode;
dadurch gekennzeichnet, dass im Schritt des Erzeugers des ausführbaren Programmcodes (**52**) ausführbare Programmcodeanteile, die ermittelten, nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes (**52**) sicherheitsmodulspezifisch angeordnet werden (TS31).

2. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass die ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes (**52**) randomisiert angeordnet werden.

3. Verfahren nach Anspruch 1 oder 2, **dadurch gekennzeichnet,** dass die ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes (**52**) derart angeordnet werden, dass sich bei Ausführung des Programmcodes (**52**) auf dem Sicherheitsmodul (**10**) ein datenträgerspezifisches Laufzeitverhalten zeigt.

4. Verfahren nach einem der Ansprüche 1 bis 3, **dadurch gekennzeichnet,** dass die ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des erzeugten Programmcodes (**52**) derart angeordnet werden, dass sich beim Speichern des Programmcodes (**52**) in einem nichtflüchtigen Speicher (**50**) des Sicherheitsmoduls (**10**) ein sicherheitsmodulspezifisches Speicherabbild ergibt.

5. Verfahren nach einem der Ansprüche 1 bis 4, **dadurch gekennzeichnet,** dass im Schritt des Ermittelns von nebenläufig ausführbaren Anteilen (B, C, D) eine Anzahl von gleichzeitig nebenläufig ausführbaren Anteilen vorgegeben wird.

6. Verfahren nach einem der Ansprüche 1 bis 5, **dadurch gekennzeichnet,** dass ausführbare Programmcodeanteile, die ermittelten, nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, in einer Tabelle angeordnet werden.

7. Verfahren nach einem der Ansprüche 1 bis 6, **dadurch gekennzeichnet,** dass als nebenläufig ausführbare Anteile (F1, F2, ..., Fk) der durch den Quellcode definierten Operationen eine Schleifenoperation (F) definierende Anteile ermittelt werden.

8. Verfahren nach einem der Ansprüche 1 bis 6, **dadurch gekennzeichnet,** dass als nebenläufig ausführbare Anteile (B, C, D) der durch den Quellcode definierten Operationen eine Rekursion definierende Anteile ermittelt werden.

9. Verfahren nach einem der Ansprüche 1 bis 8, **dadurch gekennzeichnet,** dass zumindest einer der ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, hinsichtlich eines vorgegebenem Parameters optimiert wird (S4).

10. Verfahren nach Anspruch 9, **dadurch gekennzeichnet,** dass verschiedene der ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, jeweils hinsichtlich verschiedener Parameter optimiert werden.

11. Verfahren nach einem der Ansprüche 1 bis 10, **dadurch gekennzeichnet,** dass der Schritt des Ermitteln von nebenläufig ausführbaren Anteilen (B, C, D) der durch den Quellcode definierten Operationen von einer Einrichtung unterstützt wird, die zum Erzeugen von auf einer Mehrkern- oder Mehrprozessorarchitektur parallel ausführbarem Programmcode eingerichtet ist.

12. Menge (**100**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) des gleichen Herstellers, wobei jeder dieser Sicherheitsmodule (**10**; **10''**, **10'''**) einen Speicher (**40**; **50**; **60**) und einen Prozessor (**30**) zum Ausführen von in dem Speicher (**50**) gespeichertem, ausführbarem Programmcode (**52'**; **52''**; **52'''**) umfasst, wobei die Menge (**100**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) zumindest zwei Gruppen (**101**; **102**; **103**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) umfasst und wobei jede dieser Gruppen (**101**; **102**; **103**) zumindest ein Sicherheitsmodul (**10**; **10''**, **10'''**) umfasst, **dadurch gekennzeichnet,** dass der jeweils in einem Speicher (**50**) eines Sicherheitsmoduls (**10**; **10''**, **10'''**) gespeicherte, ausführbare Programmcode (**52'**; **52''**; **52'''**) hinsichtlich dessen Funktionalität für jedes Sicherheitsmodul (**10**; **10''**, **10'''**) der Menge (**100**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) identisch ist, sich jedoch hinsichtlich dessen Laufzeitverhalten für jeweils zwei Sicherheitsmodul (**10**; **10''**) aus unterschiedlichen der Gruppen (**101**; **102**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) unterscheidet.

13. Menge (**100**) von Sicherheitsmodulen (**10**; **10''**, **10'''**) nach Anspruch 12, **dadurch gekennzeichnet,** dass das unterschiedliche Laufzeitverhalten des Programmcodes (**52'**; **52''**) für zwei Sicherheitsmodule (**10**; **10''**) aus unterschiedlichen Gruppen (**101**; **102**) darauf beruht, dass sich der Programmcode (**52'**) eines Sicherheitsmoduls (**10**) einer ersten Gruppe (**101**) von dem Programmcode (**52''**) eines Sicherheitsmoduls (**10''**) einer zweiten Gruppe (**102**) darin unterscheidet, dass beim Erzeugen des ausführbaren Programmcodes (**52'**; **52''**) für die Sicherheitsmodule (**10**; **10''**) jeweils solche ausführbaren Programmcodeanteile, die nebenläufig ausführbaren Anteilen (B, C, D) entsprechen, für eine sequenzielle Ausführbarkeit des ausführbaren Programmcodes (**52'**; **52''**) durch den Prozessor (**30**) des jeweiligen Sicherheitsmoduls (**10**; **10''**) gruppenspezifisch angeordnet worden sind.

14. System zur Erzeugung eines auf einem Sicherheitsmodul (**10**) sequentiell ausführbaren Programmcodes (**52**)
– ein Werkzeug zum Erzeugen des sequentiell ausführbaren Programmcodes (**52**) für einen Quellcode; gekennzeichnet durch
– ein Werkzeug zur Ermittlung von nebenläufig ausführbaren Anteilen (B, C, D) der durch den Quellcode definierten Operationen;
wobei das Werkzeug zum Erzeugen des Programmcodes eingerichtet ist, die von dem Werkzeug zur Ermittlung ermittelten nebenläufig ausführbaren Programmcodeanteile (B, C, D) für eine sequenzielle Ausführung auf dem Sicherheitsmodul (**10**) bereitzustellen.

15. System nach Anspruch 14 **dadurch gekennzeichnet,** dass das Ermittlungswerkzeug die ermittelten nebenläufig ausführbaren Programmcodeanteile (B, C, D) für eine nebenläufige Ausführung aufbereitet und das Erzeugungswerkzeug die aufbereiteten Programmcodeanteile (B, C, D) verwendet.

16. System nach Anspruch 14 oder 15 **dadurch gekennzeichnet,** dass das Erzeugungswerkzeug ein Monoprozessor- und/ oder Monotasking-Werkzeug ist und das Ermittlungswerkzeug ein Mehrprozessor- und/oder Multitasking-Werkzeug ist und das System zur Ausführung eines Verfahrens nach einem der Ansprüche 1 bis 11 angepasst ist.

Es folgen 3 Seiten Zeichnungen

Anhängende Zeichnungen

FIG 1

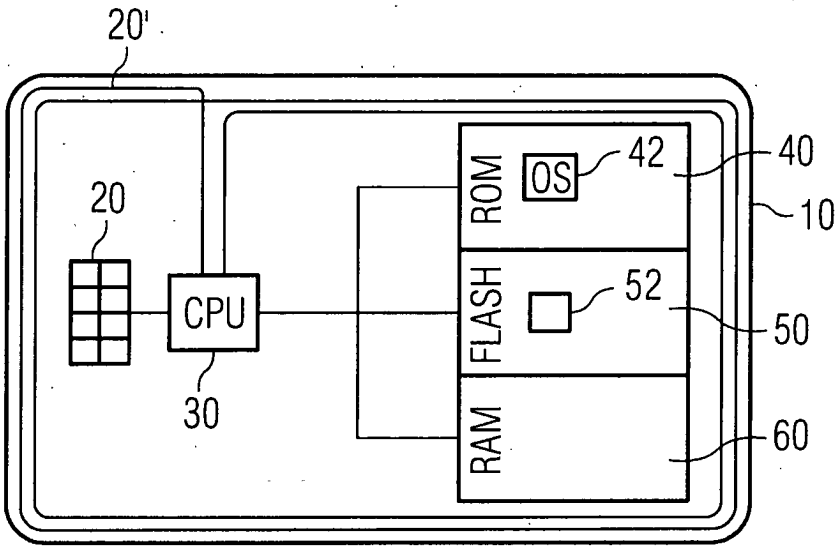


FIG 2

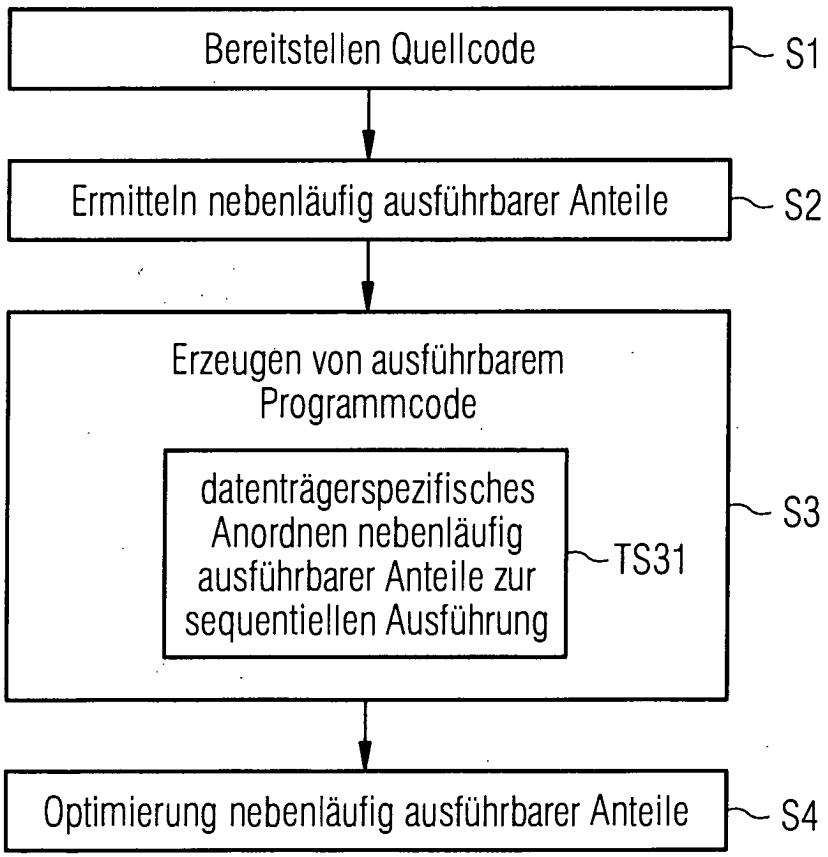


FIG 3E

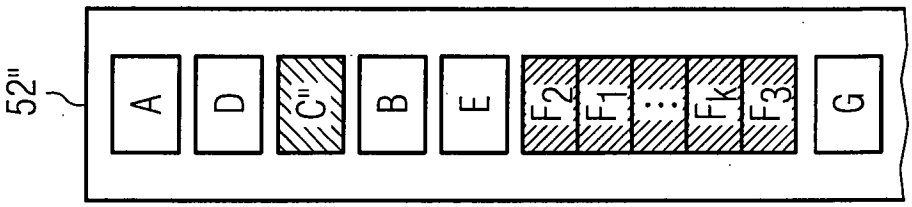


FIG 3D

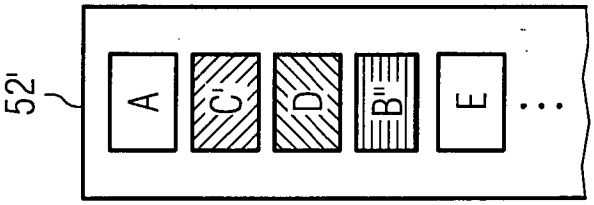


FIG 3C

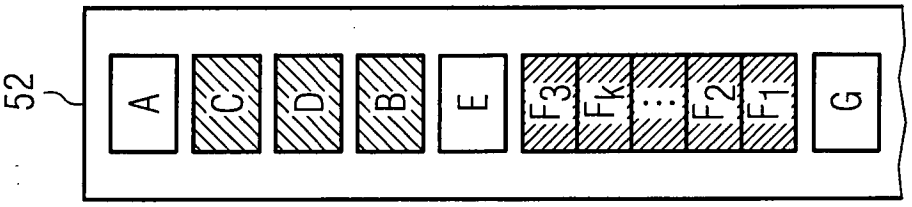


FIG 3B

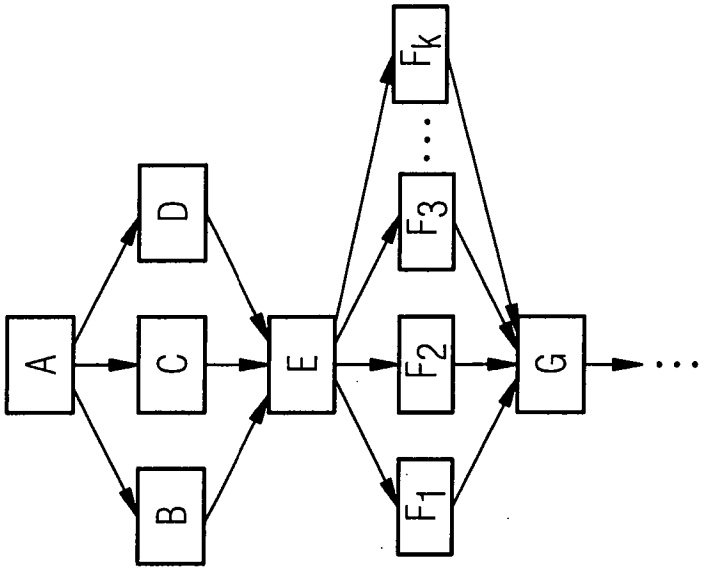


FIG 3A

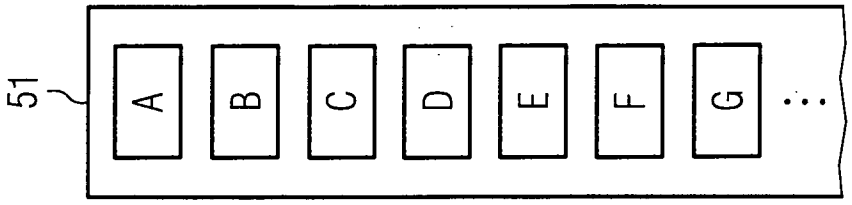


FIG 4