(19) österreichisches patentamt

(10) **AT 508 441 A2 2011-01-15**

(12) Österreichische Patentanmeldung

(21) Anmeldenummer: **A 932/2010**
(22) Anmeldetag: **08.06.2010**
(43) Veröffentlicht am: **15.01.2011**

(51) Int. Cl.[8]: ***G06K 19/077*** (200.06),
B42D 15/10 (2006.01),
G06K 19/10 (2006.01),
G06F 11/36 (2006.01)

(30) Priorität:

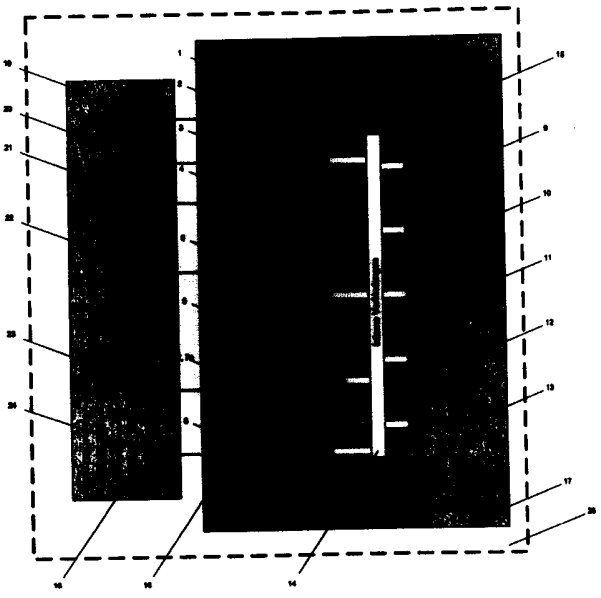
13.06.2009 DE 102009024768
beansprucht.

(73) Patentinhaber:

AUSTRIA CARD GMBH
A-1230 WIEN (AT)

(54) VERFAHREN ZUM PRÜFEN EINER CHIPKARTE DURCH SIMULATION VON ANGRIFFEN

(57) Verfahren zum Prüfen einer Chipkarte mit Angriffen auf das Betriebssystem oder die Applikationen bei der Herstellung der Chipkarte, wobei ein hardwaremäßiger Angriff auf die Chipkarte dadurch simuliert wird, dass stattdessen mit einer Angriffssoftware (18) in den Programmablauf des Betriebssystems oder der Applikationen über die Schnittstelle (16) eingegriffen wird und gezielt bestimmte Daten entsprechend einem hardwaremäßigen Angriff verändert werden, wobei Betriebssystem oder die Applikationen auf einem durch einen Chip-Simulator (17) simulierten Chip (15) ausgeführt werden.



AT 508 441 A2 2011-01-15

Zusammenfassung

Verfahren zum Prüfen einer Chipkarte mit Angriffen auf das Betriebssystem oder die Applikationen bei der Herstellung der Chipkarte, wobei ein hardwaremäßiger Angriff auf die Chipkarte dadurch simuliert wird, dass stattdessen mit einer Angriffssoftware (18) in den Programmablauf des Betriebssystems oder der Applikationen über die Schnittstelle (16) eingegriffen wird und gezielt bestimmte Daten entsprechend einem hardwaremäßigen Angriff verändert werden, wobei Betriebssystem oder die Applikationen auf einem durch einen Chip-Simulator (17) simulierten Chip (15) ausgeführt werden..

Verfahren zum Prüfen einer Chipkarte durch Simulation von Angriffen

Chipkarten, die von einem Hersteller an einen Chipkarten-Verwender geliefert werden, werden von dem Chipkarten-Hersteller mit einer bestimmten Software programmiert. Es handelt sich hierbei um ein Betriebssystem und Applikationen. Das Betriebssystem und die Applikationen enthalten verschiedene Prüfroutinen zur Überprüfung der Daten- und Ablaufintegrität der Chipkarte.

Solche fertig programmierten und ablauffähigen Chipkarten werden in einem Prüflabor mit bekannten Verfahren geprüft, wobei auch Angriffe auf die Chipkarte mit Hilfe von Laserstrahlen, Licht, ionisierender Strahlung und mit Spannungs- und Frequenzveränderungen an den Kontakten des Chips sowie physikalische Manipulationen mittels sehr feine Sonden / Kontakten und chemische Manipulationen durchgeführt werden.

Eine bekannte Angriffsmethode besteht darin, dass versucht wird, in der auf der Chipkarte ablaufenden Software gezielt Fehler auszulösen (als „fault induction" oder Fehlerinduktion bekannt).

Hierbei ist es bekannt, mit einem Laserstrahl über ein Mikroskop auf die Oberfläche des Chips einzustrahlen und hierbei in einem sehr feinen Raster den Chip der Chipkarte zu beschießen.

Solche Angriffe werden mit umfangreichen Ortsveränderungen (X, Y Koordinate auf der Chipoberfläche) des Laserstrahls durchgeführt wobei sowohl die Intensität des Lasers, die Fokussierung des Lasers, die Wellenlänge des Lasers, die Zeitpunkte des Angriffs und andere Parameter geändert werden, um einen – nicht erwünschten – Fehler auf dem Chip zu provozieren.

Insgesamt ist dies ein sehr zeitaufwendiger und ressourcenaufwendiger Prozess, der im Prinzip immer nur das Ziel hat, eine bestimmte unbekannte Stelle in der Hardware zu manipulieren. Das Endergebnis eines solchen Angriffes ist im Erfolgsfall, dass

- der Inhalt einer Speicherzelle (im flüchtigen oder nicht-flüchtigen Speicherbereich) geändert wird oder
- der Inhalt eines Prozessor-Registers (eines Haupt- oder Co-Prozessors oder ein spezielles Hardware-Register) geändert wird. Handelt es sich hierbei beispielsweise um den so genannten Programm Counter, welcher den als nächsten auszuführenden Befehl bezeichnet, so wird das Programm an einer anderen Stelle fortgesetzt.

Eine solche Veränderung des Inhaltes kann grundsätzlich persistent oder transient sein. Folgende Fälle sind zu unterscheiden:

- persistente Veränderung im nicht flüchtigen Speicher: alle nachfolgenden Lese-Zugriffe liefern den geänderten Wert zurück.
- Persistente Veränderung im flüchtigen Speicher: alle nachfolgenden Lese-Zugriffe bis zum nächsten Reset des Chips liefern den veränderten Wert zurück (nach dem Reset ist der Inhalt des flüchtigen Speichers im Allgemeinen undefiniert).
- Persistente Veränderung eines Prozessor-Registers: alle nachfolgenden Lese-Zugriffe bis zu dem Zeitpunkt an dem ein neuer Wert in das Register geschrieben wird (das kann auch selbsttägig von der Chip-Hardware verursacht werden), liefern den geänderten Wert zurück
- Transiente Veränderung einer Speicherzelle oder eines Registers: nur der aktuelle Lese-Zugriff auf die Speicherzelle oder ein Register liefert einen geänderten Wert zurück. Alle nachfolgenden Lese-Zugriffe liefern wieder den korrekten Wert zurück. Dieser Angriff funktioniert im Allgemeinen nur dann, wenn der Angriff exakt zum Zeitpunkt des Lese-Zugriffs ausgeführt wird. Es handelt sich hier um den Angriff mit der derzeit höchsten – da auf realer Hardware am einfachsten ausführbaren - Erfolgsrate.

Die derzeit bekannten Angriffe auf reale Hardware haben gemeinsam, dass das Setzen einer Speicherzelle auf einen bestimmten Wert im Allgemeinen sehr schwierig ist, sondern ein zumeist ein – von außen für den Angreifer aufgrund von Speicherverschlüsselungsmechanismen – nicht vorhersehbares Bit-Muster

gesetzt wird oder alle Bits der zu verändernden Speicherzelle auf den Wert Null oder alle Bits auf den Wert Eins gesetzt werden.

Der Erfindung liegt deshalb die Aufgabe zu Grunde, einen Angriff auf die Chipkarte und insbesondere auf den Chip der Chipkarte durch eine Software zu simulieren, ohne dass es des hardwaremäßigen Einsatzes eines Lasers oder anderer energiereicher Strahlen, einer physikalischen oder chemischen Manipulation des Chips bedarf.

Zur Lösung der gestellten Aufgabe ist die Erfindung durch die technische Lehre des Anspruches 1 gekennzeichnet.

Wesentliches Merkmal ist, dass ein hardwaremäßiger Angriff auf die Chipkarte dadurch simuliert wird, dass mit einer Angriffssoftware in den Programmablauf des Betriebssystems oder der Applikationen durch einen Chip-Simulator eines simulierten Chips über die Schnittstelle eingegriffen wird und gezielt bestimmte Daten entsprechend einem hardwaremäßigen Angriff verändert werden.

Die Idee ist jetzt, diese Angriffe durch eine Software zu simulieren, um diesen zeit- und ressourcenintensiven Hardwareprozess überhaupt nicht mehr zu benötigen und / oder eine Gefahrenanalyse durch die Simulation zu erhalten. Diese Simulation ist relativ einfach, weil das Fehlermodell welches dahintersteht einfach ist. Das Fehlermodel besteht darin, entweder den Inhalt einer Speicherzelle oder eines Registers permanent oder transient zu verändern.
Ein Vorteil ist, dass Angriffe dadurch gezielt simuliert werden können. Dazu macht man es sich zu Nutze, dass bei der Simulation die Aufteilung des logischen Speicheradressraumes in unterschiedliche physikalische Speicher (beispielsweise ROM, EEPROM, RAM, Flash) und die unterschiedliche Arten von Registern bekannt sind. So können unterschiedliche Klassen von Fehlern getrennt simuliert und betrachtet werden, wie beispielsweise:

- Veränderung des Programm-Counters
- Veränderung des Prozessor-Stacks
- Veränderung von Hardware-Registern
- Veränderungen im flüchtigen Speicher

- Veränderungen im nicht flüchtigen Speicher.

Ferner können so durch Einbringen von transienten bzw. persistenten Fehlern unterschiedliche Hardware-Angriffe simuliert werden.

Dies ist deshalb interessant, da für diese unterschiedlichen Fehlerklassen im Allgemeinen unterschiedliche Sicherheitsmaßnahmen implementiert sind.

Ebenso wird man für alle diese Klassen transiente und persistente Fehler getrennt betrachten und analysieren können.

Ob alle Fehlerarten betrachtet werden oder nur bestimmte Fehlerarten, hängt davon ab, welche Fehlerarten nach dem Stand der Technik auf der jeweiligen Chip-Hardware Plattform bekannterweise mit hoher bzw. geringer Wahrscheinlichkeit erfolgreich ausgeführt werden können.

Ein weiterer Vorteil ist im Folgenden beschrieben: Bei der tatsächlichen Durchführung von Angriffen besteht das weitere Problem, dass es auch Hardware-Gegenmaßnahmen gibt, die eben z. B. ebenfalls die Speicherintegrität oder die Integrität von Registern prüfen oder die versuchen Angriffe mittels Sensoren zu erkennen. Diese Hardware-Gegenmaßnahmen kann man auf der Softwareseite erfindungsgemäß im Simulationsfall ausschalten und getrennt betrachten, dadurch kann erfindungsgemäß der statistische oder diskrete Prozess des Angreifens der Hardware durch eine softwaremäßige Simulation wesentlich genauer betrachtet werden; er ist schneller und benötigt weniger Ressourcen schonender.

Wenn man nun erfindungsgemäß einen Angriff simulieren will, benötigt man hierzu einen softwaremäßig hergestellten Chip-Simulator mit einer Schnittstelle zur erfindungsgemäßen Angriffs-Software.

Bei einer weiteren Ausführungsform kann anstelle der Chip-Simulation Software auch eine Chip-Emulator Hardware eingesetzt werden.

Es ist bereits bekannt, bei Software-Anwendungen und bei der Programmierung der Software eines Betriebssystems für eine Chipkarte einen (Chip-)Simulator zu verwenden. Bisher ist es jedoch noch nicht bekannt, einen solchen Simulator für die Simulation von Angriffen auf das Betriebssystem der Chipkarte gezielt einzusetzen.

Die Schnittstelle zwischen Chip-Simulator und Angriffs-Software ist relativ einfach. Zumindest folgende Zugriffsmöglichkeiten werden benötigt:

- Zugriff auf die Speicherzellen des simulierten Chips
- Zugriff auf die Register (aller enthaltenen Prozessoren) des simulierten Chips

Die Zugriffe müssen zu jedem Zeitpunkt schreibend und lesend möglich sein. Heute bekannte Chip-Simulatoren bieten im Allgemeinen diese Möglichkeiten.

Über diesen einfachen Mechanismus kann man nun erfindungsgemäß alle Arten von Angriffen simulieren.

Ein Angriff kann beispielsweise wie folgt simuliert werden:
Die Angriffs-Software prüft durch Auslesen des Programm-Counters bzw. durch Zählen der abgearbeiteten Prozessor-Instruktionen, an welcher Stelle im Programmablauf sich die Chip-Software gerade befindet. Stimmt diese mit der anzugreifenden Stelle überein, wird ein Angriff durchgeführt. Durch Variieren der anzugreifenden Stelle können Variationen des Zeitpunktes des Angriffes simuliert werden. Der Angriff selbst manipuliert nun einen bestimmten Speicherbereich im flüchtigen oder nicht flüchtigen Speicher-Bereich oder von Registern. Durch die Variation des angegriffenen Speicherbereiches bzw. der unterschiedlichen Register werden daher die unterschiedlichen Orte des Angriffes an der Chipoberfläche (X, Y Koordinaten) simuliert.
Durch Variation der Anzahl der veränderten Speicher-Bits und der geschriebenen Werte d.h. unterschiedlicher Angriffsparameter (insbesondere Fokussierung, Energie, Wellenlänge der eingestrahlten elektromagnetischen Wellen, Angriff durch direkte Kontaktierung mit Micro-Probes, Angriff durch ionisierende Strahlung) kann beispielsweise die Angriffsart simuliert werden.

Man kann deshalb im Wesentlichen auf die im Labor durchgeführten hardwaremäßigen Angriffe verzichten, die außerordentlich zeitaufwendig sind und Maschinen-Ressourcen benötigen. Nachteil dieser hardwaremäßigen Angriffe ist nämlich, dass man aus physikalischen oder zeitlichen Gründen nicht jede Stelle des Chips bei gleichzeitiger Variation aller anderen Parameter beschießen kann, weil dies entweder aus räumlichen Gründen nicht möglich ist oder sich eine außerordentliche zeitaufwendige Überprüfung ergeben würde.

Vorteil der Erfindung ist, dass man Testergebnisse, die beim softwaremäßigen Angriff auf den Chip erkannt werden, bei der Programmierung des Chips oder der Chip-Betriebssoftware wieder berücksichtigt werden können. D.h., bei der Chipentwicklung können Ergebnisse bezüglich der Angreifbarkeit des Chips erkannt und vorweggenommen werden, wodurch die Chipsoftware insgesamt verlässlicher und sicherer wird. Weiters würden damit die Kosten einer Sicherheitszertifizierung wesentlich gesenkt und der Zeitaufwand dementsprechend minimiert werden.

Solche Angriffe, die auf der Hardwareseite nach dem Stand der Technik stattfinden, werden im Zeitablauf Wochen verwendet, um solche Angriffe realistisch durchzuführen, was mit den Merkmalen der vorliegenden Erfindung mit Sicherheit vermieden wird. Dies ist sowohl ein Vorteil für den Entwickler (Austria Card) als auch für das Prüflabor, weil einerseits der Entwicklungsaufwand minimiert wird und andererseits die Prüfroutinen und die Prüfprozeduren wesentlich verkürzt werden.

Oft müssen sicherheitskritische Systeme in regelmäßigen Abständen (z.B. zwei Jahre) zertifiziert werden und es sind immer neue Zertifikationsprozesse notwendig, die mit den Merkmalen der vorliegenden Erfindung damit wesentlich verkürzt und kostengünstiger gestaltet werden können.

Im Folgenden ist der typische Ablauf der Kommando-Abarbeitung innerhalb eines Chipkarten-Betriebssystems beschrieben.

Die typische Kommando-Abarbeitung innerhalb des Chipkarten-Betriebssystems läuft wie folgt ab: Alle Kommandos an die Chipkarte empfängt diese über eine I/O-Schnittstelle. Fehlererkennungs- und –Korrekturmechanismen führt der I/O-Manager bei Bedarf völlig unabhängig von den übrigen, darauf aufbauenden Schichten aus. Nachdem ein Kommando vollständig und fehlerfrei empfangen wurde, muss der Secure Messaging Manager diesen gegebenenfalls entschlüsseln oder auf Integrität prüfen. Findet eine gesicherte Datenübertragung statt, ist dieser Manager sowohl für Kommando als auch Antwort völlig transparent.

Nach dieser Bearbeitung versucht die darüberliegende Schicht, der Kommandointerpreter, das Kommando zu dekodieren. Ist dies nicht möglich, folgt ein Aufruf des Returncode-Managers, welcher einen entsprechenden Returncode generiert und via I/O-Manager an das Terminal zurücksendet. Es kann notwendig sein, den Returncode-Manager applikationsspezifisch zu gestalten, da die Returncodes nicht zwangsläufig für alle Anwendungen einheitlich sind. Konnte das Kommando jedoch dekodiert werden, ermittelt der Logical Channel Manager den angewählten Kanal, schaltet auf dessen Zustände um und ruft dann im Gutfall den Zustandsautomaten auf.

Dieser prüft nun, ob das Kommando an die Chipkarte mit den gesetzten Parametern im aktuellen Zustand überhaupt erlaubt ist. Ist dies der Fall, wird der eigentliche Programmcode des Anwendungskommandos ausgeführt, welcher die Abarbeitung des Kommandos übernimmt. Falls das Kommando im aktuellen Zustand verboten ist oder die Parameter dazu nicht erlaubt sind, erhält das Terminal über Returncode Manager und I/O-Manager eine entsprechende Meldung.

Die Dateiverwaltung selber benutzt einen weiteren Speichermanager, der die komplette Verwaltung des physikalisch adressierten nichtflüchtigen Speichers übernimmt. Damit ist sichergestellt, dass nur in diesem Programmmodul mit echten physikalischen Adressen gearbeitet wird, was die Portabilität und Sicherheit des Betriebssystems erheblich steigert.

Die beigefügte Abbildung zeigt ein Blockschaltbild, in welchem einerseits der durch die Chip-Simulationssoftware (17) simulierte Chip (15) und die typischen den Chip aufbauenden Komponenten (1) – (14) sowie weiters die erfindungsgemäße Angriffssoftware (18) eingezeichnet sind. Diese führt Zugriffe (19) – (24) über die bereits bekannte Schnittstelle (16) auf folgende Komponenten des simulierten Chips (15) zu:

- Speichereinheit (1) bestehend aus
  - RAM (2): Zugriff (19)
  - EEPROM (3): Zugriff (20)
  - ROM (4): Zugriff (21)
- CPU (5) enthaltend
  - Register (6): Zugriff (22)
- Beliebige Anzahl von Co-Prozessoren (7): Zugriff (23)
- Special Function Register (8): Zugriff (24)

Auf die Komponenten

- Timer (9)
- Takterzeuger (10)
- I/O Baustein (11)
- Sensoren (12)
- RNG (13)

wird typischerweise durch Zugriff auf spezielle definierte Speicherbereiche (d.h. auf (2), (3), (4) und Special Function Register (8) zugegriffen.

Mit den eingezeichneten Zugriffen (19) – (24) können somit gezielt Daten gelesen und verändert werden, um einen Fehlerfall zu provozieren.

Insbesondere kann ein Speicherwert innerhalb der Speichereinheit (1) oder ein Register (6) innerhalb der CPU (5) oder ein Wert innerhalb eines Co-Prozessors (7) oder ein Special Function Register (8) verändert werden.

Die Chip-Simulationssoftware (17) und die Angriffssoftware (18) werden in einer Laufzeitumgebung (25) beispielsweise einem Computer (oder mehreren) betrieben.

Zeichnungslegende

1 Speichereinheit
2 RAM
3 EEPROM
4 ROM
5 CPU
6 Register
7 ,7a Co-Prozessoren (beliebige Anzahl)
8 Special Function Register
9 Timer
10 Takterzeuger
11 I/O Baustein
12 Sensoren
13 RNG
14 Interne Verbindungen (Datenbusse)
15 Simulierter Chip
16 Schnittstelle des Chip-Simulators
17 Chip-Simulator
18 Angriffssoftware
19 Zugriff auf RAM
20 Zugriff auf EEPROM
21 Zugriff auf ROM
22 Zugriff auf Register
23 Zugriff auf Co-Prozessoren
24 Zugriff auf Special Function Register
25 Laufzeitumgebung von Chip-Simulationssoftware und Angriffssoftware

Patentansprüche

1. Verfahren zum Prüfen einer Chipkarte mit Angriffen auf das Betriebssystem oder die Applikationen bei der Herstellung der Chipkarte, **dadurch gekennzeichnet, dass** ein hardwaremäßiger Angriff auf die Chipkarte dadurch simuliert wird, dass mit einer Angriffssoftware (18) in den Programmablauf des Betriebssystems oder der Applikationen durch einen Chip-Simulator (17) eines simulierten Chips (15) über die Schnittstelle (16) eingegriffen wird und gezielt bestimmte Daten entsprechend einem hardwaremäßigen Angriff verändert werden.

2. Verfahren nach Anspruch 1, **dadurch gekennzeichnet, dass** insbesondere ein Speicherwert innerhalb der Speichereinheit (1) oder ein Register (6) innerhalb der CPU (5) oder ein Wert innerhalb eines Co-Prozessors (7) oder ein Special Function Register (8) verändert wird.

3. Verfahren nach Anspruch 1 oder 2, **dadurch gekennzeichnet, dass** durch Variation der angegriffenen Speicherzelle unterschiedliche Orte des Angriffes auf der Chipoberfläche (X, Y Koordinaten) simuliert werden.

4. Verfahren nach einem der Ansprüche 1 bis 3, **dadurch gekennzeichnet, dass** durch Auslesen des Programm-Counters bzw. zählen der abgearbeiteten Prozessor-Instruktionen unterschiedliche Zeitpunkte des Angriffes simuliert werden.

5. Verfahren nach einem der Ansprüche 1 bis 4, **dadurch gekennzeichnet, dass** durch Einbringen von transienten bzw. persistenten Fehlern unterschiedliche Hardware-Angriffe simuliert werden.

6. Verfahren nach einem der Ansprüche 1 bis 5, **dadurch gekennzeichnet, dass** durch Variation der Anzahl der veränderten Speicher-Bits bzw. Bytes und der geschriebenen Werte unterschiedliche Angriffsparameter (insbesondere Fokussierung, Energie, Wellenlänge der eingestrahlten elektromagnetischen Wellen, Angriff durch direkte Kontaktierung mit Micro-Probes, Angriff durch ionisierende Strahlung) simuliert werden.

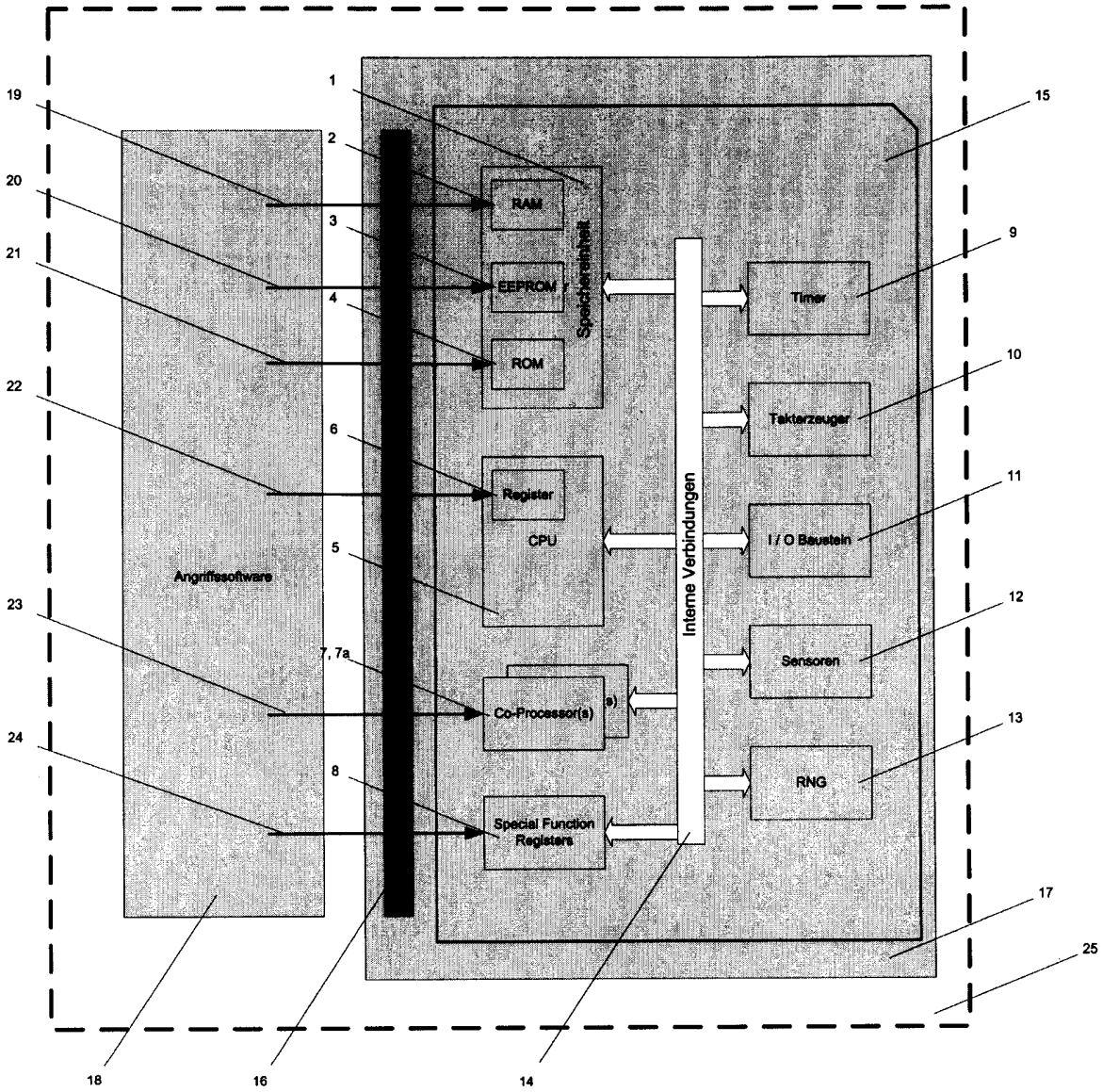
7. Verfahren nach einem der Ansprüche 1 bis 6, **dadurch gekennzeichnet, dass** durch diskrete oder statistische Auswertung der Simulationsergebnisse die Wirksamkeit von unterschiedlichen im Betriebssystem oder den Applikationen implementierten Gegenmaßnahmen bewertet wird.

8. Verfahren nach einem der Ansprüche 1 bis 7, **dadurch gekennzeichnet, dass** die Angriffssoftware (18) für die Provokation von Angriffen auf das Betriebssystem oder die Applikationen der Chipkarte verwendet wird.

9. Verfahren nach einem der Ansprüche 1 bis 8, **dadurch gekennzeichnet, dass** anstelle der Chip-Simulator Software (17) eine Chip-Emulator Hardware benutzt wird.

10. Angriffssoftware zur Ausführung des Verfahrens nach einem der Ansprüche 1 bis 9, **dadurch gekennzeichnet, dass** die Angriffssoftware (18) innerhalb des Chipkartenbetriebssystems oder der Applikationen eingebettet ist.

11. Chipkarte, **dadurch gekennzeichnet, dass** diese ein Betriebssystem oder Applikationen enthält, welche(s) während oder nach der Entwicklung mit einem Verfahren nach einem oder mehreren der Ansprüche 1 bis 10 geprüft wurde(n).


19
1
15
2
20
RAM
3
9
21
EEPROM
Timer
4
ROM
10
22
Tekterzeuger
6
Register
11
CPU
I / O Baustein
Angriffssoftware
5
12
23
Sensoren
7, 7a
Co-Processor(s)
13
24
RNG
8
Special Function Registers
17
25
18
16
14