(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
3. August 2017 (03.08.2017)

WIPO | PCT

(10) Internationale Veröffentlichungsnummer
WO 2017/129184 A1

(51) Internationale Patentklassifikation:
G06F 21/31 (2013.01) *G06F 21/85* (2013.01)
G06F 21/55 (2013.01) *H04L 9/00* (2006.01)
G06F 21/60 (2013.01) *H04L 29/06* (2006.01)
G06F 21/62 (2013.01) *H04L 9/08* (2006.01)
G06F 21/71 (2013.01)

(21) Internationales Aktenzeichen: PCT/DE2017/200002

(22) Internationales Anmeldedatum:
17. Januar 2017 (17.01.2017)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:
10 2016 000 328.6
18. Januar 2016 (18.01.2016) DE

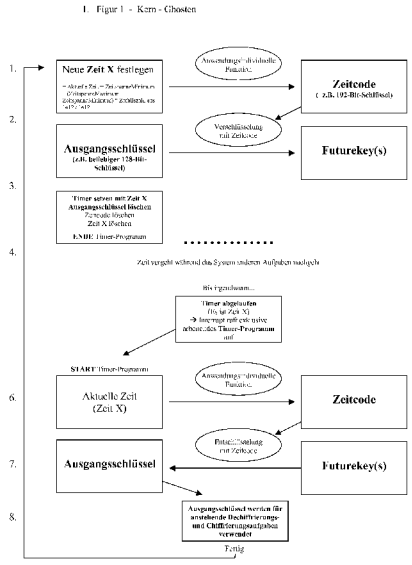
(72) Erfinder; und
(71) Anmelder : **HARRAS, Roland** [DE/DE]; Roland Harras, Harthauser Str. 68, 81545 München (DE).

(81) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare nationale Schutzrechtsart)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY,

[Fortsetzung auf der nächsten Seite]

(54) **Title**: METHOD FOR SAVING DATA WITH MULTI-LAYER PROTECTION, IN PARTICULAR LOG-ON DATA AND PASSWORDS

(54) **Bezeichnung** : VERFAHREN ZUR MEHRSCHICHTIG GESCHÜTZTEN SICHERUNG VON DATEN INSBESONDERE ANMELDEDATEN UND PASSWÖRTERN



Figur... Figure
Kern-Ghosten... Core ghosting
Neue Zeit X festlegen... Stipulate new time X
Aktuelle Zeit + ZeitspanneMinimum + (ZeitspanneMaximum - ZeitspanneMinimum) * Zufallszahl aus 1e12/1e12... Current time + minimum period + (maximum period - minimum period) * random number from 1e12/1e12
Anwendungsindividuelle Funktion... Application-specific function
Zeitcode (=z.B. 192-Bit-Schlüssel)... Time code (=e.g. 192-bit key)
Ausgangsschlüssel (z.B. beliebiger 128-Bit-Schlüssel)... Output key (e.g. any desired 128-bit key)
Verschlüsselung mit Zeitcode... Encryption with time code
Timer setzen mit Zeit X... Set timer with time X
Ausgangsschlüssel löschen... Erase output key
Zeitcode löschen... Erase time code
Zeit X löschen... Erase time X
ENDE Timer-Programm... END of timer program
Zeit vergeht während das System anderen Aufgaben nachgeht... Time elapses while the system is following up other tasks
Bis irgendwann ... Until
Timer abgelaufen... Timer expired
(Es ist Zeit X)... (It is time X)
Interrupt ruft exklusive arbeitendes Timer-Programm auf... Interrupt calls exclusively operating timer program
START Timer-Programm... START of timer program
Aktuelle Zeit... Current time
(Zeit X)... (Time X)
Ausgangsschlüssel... Output key
Entschlüsselung mit Zeitcode... Decryption with time code
Zeitcode... Time code
Ausgangsschlüssel werden für anstehende Dechiffrierungs- und Chiffrierungsaufgaben verwendet... Output keys are used for pending decoding and coding tasks
Fertig... Complete

(57) **Abstract**: Almost every month, there are new reports of hackers who were able to acquire millions of data items and passwords. The problem: even if data are sufficiently encrypted, there must be a key somewhere for decryption. If this key can be stolen, the best encryption is of no use. Instead of conventional keys, the present overall data protection concept uses future events as a secret basis for encryption. Data are coded repeatedly with variable and partly only transient keys which are not permanently stored but are coded with time codes which result from unpredictable future timer events and are therefore impossible to steal. Various measures protect keys against viewing even during the immediate use thereof, and an optional hardware expansion excludes any possibilities of manipulation, with the result that there is no longer any risk even in captured systems.

(57) **Zusammenfassung**:

[Fortsetzung auf der nächsten Seite]

WO 2017/129184 A1

MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare regionale Schutzrechtsart)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), eurasisches (AM, AZ, BY, KG, KZ, RU, TJ, TM), europäisches (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Erklärungen gemäß Regel 4.17:

— *Erfindererklärung (Regel 4.17 Ziffer iv)*

Veröffentlicht:

— *mit internationalem Recherchenbericht (Artikel 21 Absatz 3)*

Fast monatlich gibt es neue Berichte von Hackern die Millionen Daten und Passwörter erbeuten konnten. Das Problem: Selbst wenn Daten ausreichend verschlüsselt sind, muss es doch irgendwo einen Schlüssel zum Entschlüsseln geben. Kann dieser entwendet werden, nützt die beste Verschlüsselung nichts. Das vorliegende Datenschutz-Gesamt-Konzept verwendet anstatt klassischer Schlüssel zukünftige Ereignisse als geheime Basis für Verschlüsselungen. Daten werden mehrmals mit variablen und teils nur kurzlebigen Schlüsseln chiffriert, die nicht dauerhaft gespeichert, sondern mit Zeitcodes chiffriert werden, welche sich aus unvorhersehbaren zukünftigen Timer-Events ergeben und somit unmöglich entwendet werden können. Verschiedene Maßnahmen sichern Schlüssel sogar während deren unmittelbaren Verwendung gegen Einsichtnahme und eine optionale Hardwareerweiterung schließt jegliche Manipulationsmöglichkeiten aus, sodass selbst bei gekaperten Systemen keine Gefahr mehr besteht.

I. Beschreibung

Verfahren zur mehrschichtig geschützten Sicherung von Daten, insbesondere Anmeldedaten und Passwörtern

Inhaltsverzeichnis

A. HINTERGRUND BZW. PROBLEMSTELLUNG (TECHN. GEBIET)

In den verschiedensten IT-Anwendungen und vor allem auf Internet-Webseiten können oder müssen sich Benutzer registrieren (Registrierung). Hierzu wird zumeist ein Benutzername (oft die eMail-Adresse) in Kombination mit einem Passwort zur Authentifizierung angegeben. Mit dieser Kombination kann der Benutzer sich später jederzeit wieder einloggen um Zugriff auf seine persönlichen Einstellungen, sein Konto oder die ihm (meist durch vorangegangene oder nachfolgende Bezahlung) zugewiesenen Dienste zu erlangen (Anmeldung/Login). Milliarden Menschen haben zum Teil mehrere so genannte Web-Accounts auf verschiedensten Websites. Allen voran Online-Shops, eBay etc, Banken, Reiseportalen, App-Anbietern usw.

Die jeweiligen (Website-) Server / Anwendungen speichern diese Anmeldedaten / Zugangsdaten (Benutzername und Kennwort) zumeist in Datenbanken entweder in Klartext oder verschlüsselt. Obwohl der Zugriff auf diese Datenbanken durch verschiedene Verfahren geschützt wird, häufen sich in den letzten Jahren und Monaten erfolgreiche Hacker-Attacken in denen massenweise solche Anmeldedaten erbeutet wurden. Aufgrund der oftmals einfachen Verschlüsselung, der Tatsache dass Passwörter meistens - wenn überhaupt - nur gehasht sind und der Masse an erbeuteten Daten lassen sich diese Daten mit entsprechend hoher Rechenleistung wieder entschlüsseln. In manchen Fällen wird der dazu notwendige Schlüssel sogar mit erbeutet. Manchmal werden die Daten jedoch auch unverschlüsselt erbeutet, weil die Server-Eigner nicht mit dem Gelingen eines solchen Hacker-Angriffs gerechnet haben. Die NSA-Affäre hat gezeigt, dass viele Bereiche unseres digitalen Lebens keineswegs so sicher sind wie wir das geglaubt haben.

Der resultierende Schaden ist nicht nur für das Vertrauen und Image der jeweiligen Website enorm; Es birgt für die Anwender ein hohes Risiko des Missbrauchs dieser Daten. Da 95% der Anwender diese Anmeldedaten in selber Form mehrfach, d.h. bei mehreren Anwendungen oder Webseiten, benutzen, geht von der Erbeutung solcher Anmeldedaten eine unbezifferbare Bedrohung aus. So kann beispielsweise ein auf einer einfach geschützten Website erbeutetes Passwort genutzt werden um im Namen des Anwenders Einkäufe auf div. Online-Shops zu machen oder Zugriff auf eine vom Anwender genutzte Cloud (Online-Speicher) zu erlangen und damit zu intimen Daten und Bildern zu gelangen oder gar der Zugang zu weitaus sensibleren Anwendungsbereichen wie dem Online-Banking erlangt werden.

Dies ist ein massives und teilweise noch stark unterschätztes Risiko unserer Zeit in der sich ein zunehmender Bereich unseres Lebens online manifestiert.

Bei der vorliegenden Erfindung handelt es sich um ein Verfahren zum Schutz von Daten vor unbefugter Einsichtnahme, welches den Anspruch hat, selbst bei Angriffen auf allerhöchstem Niveau, unknackbar zu sein.

B. KERN DES VERFAHRENS

Kern des Verfahrens ist die Lösung des Hauptproblems jedes noch so guten Datenschutz- bzw. Verschlüsselungsverfahren: Wie versteckt man die Schlüssel. Das ist vergleichbar mit einem versteckten Schatz. Das beste Versteck ist nur so gut wie geheim die Hinweise darauf sind. Aber irgendwer muss letztlich wissen wo der Schatz versteckt ist; Und sei es dass diese Schatzkarte auf mehrere Personen oder Stellen aufgeteilt ist. So ist es auch mit dem Schlüssel von verschlüsselten Daten. Dieser Schlüssel muss irgendwo stehen und die Anwendung die mit diesen Daten arbeitet muss darauf zugreifen können. Also kann auch ein anderes Programm, wie z.B. ein Schadprogramm (Virus, Bootkits), darauf zugreifen. Es muss nur wissen wie die reguläre DV-Anwendung auf den oder die Schlüssel zugreift und sich die entsprechenden Rechte verschaffen. Also, wo und wie sie versteckt werden. Und, - um dies heraus zu finden gibt es mehrere Möglichkeiten.
Somit ist jede Daten-Verschlüsselung potentiell gefährdet sobald sich ein Angreifer Zugriff auf das jeweilige System verschaffen kann. Es macht von daher zumeist auch gar keinen Sinn, Daten noch intensiver oder mehrfach zu verschlüsseln da der Schwachpunkt bleibt: Die Schlüssel können mit genug Arbeits-, Zeit- und Geld-Einsatz immer gefunden und entwendet werden.

Aus diesem Grund konzentriert man sich in der Vergangenheit auf Schutzmechanismen um einen solchen Zugriff zu verhindern. Doch wie vielfach gezeigt wurde, ist trotz aufwendiger Firewalls, Viren-Scanner und noch so ausgeklügelter Sicherheitstechnik ein 100%-iger Schutz gegen das Eindringen eines Angreifers nicht möglich.

Die vorliegende Erfindung löst dieses Problem indem es die verwendeten Schlüssel (Ausgangsschlüssel) ebenfalls verschlüsselt, woraus sog. Futurekeys entstehen. Doch damit allein hätte man wieder dasselbe Problem. Wohin mit dem Schlüssel der den/die Ausgangsschlüssel verschlüsselt hat? Aus diesem Grund wird dieser Schlüssel aus einem Zeitcode auf Basis einer in der Zukunft liegenden Systemzeit (Zeit X) generiert und ein Timer

gesetzt welcher zu dieser zukünftigen Zeit X (z.B. über einen sog. Interrupt, idealerweise einem NMI (Not Maskable Interrupt)) ein Programm (das Timer-Programm) aufruft, das die Entschlüsselung der Futurekeys zu dieser Zeit X mit dem aus der dann gegebenen Systemzeit generierbaren Zeitcode wieder ermöglicht. Es stehen dann die Ausgangsschlüssel wieder zur Verfügung obwohl sie in der Zwischenzeit nur verschlüsselt vorlagen ohne dass der dazu verwendete Schlüssel existierte.

Also (Siehe Figur 1):

1. Zukünftige Zeit X → (anwendungsindividuelle Funktion) → Zeitcode
2. Ausgangsschlüssel → (Verschlüsselungsverfahren xy mit Zeitcode als Schlüssel) → Futurekey(s)
3. Ausgangsschlüssel löschen, Timer setzen, Timer-Programm beenden
4. ..... System arbeitet normal weiter
5. Zeit X ist erreicht. Timer → (Interrupt) → Timer-Programm (erneut)
6. Zeit X → Zeitcode
7. Futurekey(s) → (Entschlüsselungsverfahren xy mit Zeitcode als Schlüssel) → Ausgangsschlüssel
8. Aufgelaufene De-/ Chiffrierungsaufgaben werden ausgeführt und dann wird bei Schritt 1 fortgefahren

Das bedeutet, die Ausgangsschlüssel stehen nur zur Verfügung während das Timer-Programm läuft. Da dieses als NMI-Programm exklusiv arbeitet könnten Schadprogramme erst danach wieder versuchen die Ausgangsschlüssel zu erbeuten, doch dann sind sie bereits wieder verschwunden (verschlüsselt und gelöscht).

Somit ergibt sich Anspruch 1 dieses Patents...

C. BESCHREIBUNG ANSPRUCH 1 (GHOSTEN)

Verfahren zur mehrschichtig geschützten Sicherung von Daten, insbesondere Anmeldedaten und Passwörtern, wobei

a) aus zur Verschlüsselung von Daten eingesetzte Schlüssel (Ausgangsschlüssel) unter Verwendung eines Zeitcodes, der aus einem in Relation zur aktuellen Systemzeit in der Zukunft

liegenden System- oder Alarm-Zeitwert (Zeit X) errechnet wird, jeweils ein so genannter Futurekey berechnet wird, und

b) anschließend der/die Ausgangsschlüssel gelöscht werden, und

c) ein Timer so gesetzt (programmiert und gestartet) wird, dass er zur Zeit X abläuft oder auslöst und damit ein Timer-Programm aufruft, welches aus der dann vorliegenden System- oder Timer-Alarm-Zeit den Zeitcode aus a) wieder generiert und damit aus den Futurekeys die in a) verwendeten Ausgangsschlüssel wieder berechnet, um damit anstehende Dechiffrierungs- und Chiffrierungsaufgaben auszuführen;

d) wobei sich das von a) bis c) beschriebene Verfahren kontinuierlich wiederholt und die jeweilige Zeit X, zu der es jeweils möglich ist die Ausgangsschlüssel aus den jeweiligen Futurekeys zu errechnen, stets direkt in einen Timer programmiert wird und gespeicherte Informationen über diese Zeit X inklusive dem Zeitcode aus allen Speichermedien außer dem Timer gelöscht werden;

e) wobei Dechiffrierungs- und Chiffrierungsaufgaben bis zur jeweils nächsten Zeit X zwischengespeichert werden und diese Aufgaben zur Zeit X von dem o.g. Timer-Programm abgearbeitet werden. (Fig. 1)

1. Ausgangsschlüssel

Unter **Ausgangsschlüssel** werden grundsätzlich jegliche kryptologischen bzw. kryptografischen Schlüssel verstanden wie sie verwendet werden um Daten zu ver- oder entschlüsseln, also von Klartext zu Geheimtext oder umgekehrt umzuwandeln, wobei hier hauptsächlich an symmetrische Schlüssel / Verschlüsselungen gedacht wird wenngleich das Verfahren ebenfalls bei asymmetrischen Schlüsseln angewendet werden kann.

Diese Schlüssel werden nun also nicht irgendwo im Computer-System auf dem die jeweilige Verschlüsselungssoftware läuft oder anderswo gespeichert bzw. versteckt, sondern selbst symmetrisch (z.B. mit AES-192, mit hoher (evtl. höchstmöglicher) Rundenzahl, mindestens 16) verschlüsselt. Das Ergebnis dieser (Ausgangs-)Schlüssel-Verschlüsselung nennen wir **Futurekey**.

2. Zeitcode, Zeit X und Formel

Zu dieser Ausgangsschlüssel-Verschlüsselung wird ein **Zeitcode** als Schlüssel verwendet. Dieser Zeitcode wird mit Hilfe einer vorzugsweise anwendungsindividuellen Funktion / Formel aus

einem zukünftigen Systemzeitwert generiert. Dazu wird aus einer anwendungsindividuellen groben Zeitspanne (beispielsweise 2,5 Sek.) und einer ebenfalls anwendungsindividuellen Mindestzeit unter Verwendung einer Zufallszahl dieser zukünftige Systemzeitwert (**Zeit X**) festgelegt. Durch Einbringung des Zufallswertes in eine nach oben und unten abgegrenzte Zeitspanne erhält diese eine gewisse Variabilität und ist somit nicht vorhersehbar. Die ebenfalls anwendungsindividuelle Formel dafür könnte vereinfacht z.B. so aussehen:
Zeit X = Aktuelle Zeit + ZeitspanneMinimum + (ZeitspanneMaximum - ZeitspanneMinimum) * Zufallszahl aus 1e12 / 1e12.

Der Zeitcode sollte eine für einen Schlüssel zeitgemäße Länge haben. Derzeit sollte er also mindestens 128 Bit haben. Empfohlen wird jedoch mindestens 192 Bit.
Da die Systemzeit i.d.R. maximal ein 64-Bit-Wert ist und ohne Datum und bei einer Zeitauflösung von 1 Mikrosekunde (1 millionstel Sekunde) gar nur 37 Bit ausreichen, bedarf es einer anwendungsindividuellen Formel um aus Zeit X einen z.B. 192-Bit-Zeitcode zu generieren. Die Länge des Zeitcodes kann festgelegt oder in einem gewissen Rahmen variabel sein. Wobei wie erwähnt 128 Bit als Untergrenze angesehen werden sollte.

3. Sicheres löschen

Unmittelbar nachdem ein Futurekey berechnet wurde, wird der dazugehörige Ausgangsschlüssel gelöscht. Wenn hier im Dokument von „gelöscht“ die Rede ist, versteht sich darunter immer „**sicher gelöscht**“. „Sicher gelöscht“ bedeutet hier und im gesamten Dokument, es werden nicht nur Zeiger/Hinweise/Gültigkeitsvermerke etc. gelöscht, sondern der für den jeweiligen Inhalt verwendete Speicher mehrfach mit unterschiedlichen Werten überschrieben. Hierfür sollte mindestens der NSA-Standard DoD-5220.22-M (ECE) verwendet werden. Es ist darauf zu achten, dass die Ausgangsschlüssel in allen Medien sicher gelöscht sind, auch in entfernten Rechnern, Arbeitsspeicher, Festplatten, Zwischenspeichern, Auslagerungsdateien, etc. Auf die Handhabung für technische Notfälle wird später eingegangen.

Im Ergebnis liegen die zur Verschlüsselung von Daten eingesetzten Schlüssel (Ausgangsschlüssel) – die Achillesferse jeder Verschlüsselung – nur in sicher verschlüsselter Form als Futurekeys vor.

Nun muss noch der zur Erzeugung der Futurekeys verwendete Schlüssel, der Zeitcode, vernichtet werden. Er muss ebenfalls sicher gelöscht werden.

4. Timer - Mindestanforderung

Bevor nun auch noch sämtliche Speicherungen und Hinweise auf die Zeit X sicher gelöscht werden, muss diese in einen **Timer** (siehe auch Anspruch 2) programmiert und dieser gestartet werden. Dann wird Zeit X ebenfalls sicher gelöscht. Nun „weiß" nur noch der Timer zu welcher Zeit aus dem dann vorliegenden Systemzeitwert oder der Alarmzeit der Zeitcode rekonstruiert werden kann um damit die Ausgangsschlüssel aus den Futurekeys zu entschlüsseln.

Essentiell dabei ist, dass der in den Timer programmierte Wert (Zeit X), welcher entweder der Startwert eines Timers ist der auf 0 zurück läuft/zählt oder der Wert einer Alarmzeit ist, sowie ein etwaiges Zeit-Register, nicht ausgelesen werden kann bzw. gegen Auslesen gesperrt ist, so lange bis die Alarm-Zeit oder Null-Zeit (Countdown) erreicht ist. Sollte der Standard-Timer des Computersystems (auf dem eine Software nach dem hier beschriebenen Verfahren arbeiten soll) dies nicht sicherstellen können, muss die später beschriebene Timer-Hardwareerweiterung verwendet werden, da eine Schadsoftware ansonsten die Zeit X aus dem Timer auslesen könnte und damit - bei Wissen der anwendungsindividuellen Formel - den Zeitcode errechnen und somit die Futurekeys entschlüsseln könnte. Die Ausführungen in den Kapiteln D. (Beschreibung ANSPRUCH 2: (Timer-Hardwareerweiterung)), J. (Fehlerkorrektur beim Start des Timer-Programms), K.2. (Geringe Timer-Auflösung und Schlüsselbreite), K.4.a) (128-Bit-Timer-Register) sind unbedingt zu beachten.

5. Interrupt / NMI

All dies (Zeit X -> Zeitcode, Ausgangsschlüssel -> Futurekeys, Löschen, Timer setzen) wird beim Start des Systems und ansonsten immer wieder am Ende des sog. Timer-Programms umgesetzt. Dieses Timer-Programm wird aufgerufen sobald der Timer abläuft bzw. Alarm auslöst und somit einen **Interrupt** auslöst, welcher seinerseits die laufenden Programme (was auch immer) unterbricht und das Timer-Programm, das in der Interrupt-Sprung-Tabelle eingetragen wurde, aufruft. Hierfür sollte ein so genannter NMI (Not Maskable Interrupt) genutzt werden, dessen unmittelbare Ausführung nicht unterbunden werden kann. (siehe auch Anspruch 2)
Zu Beginn des Timer-Programms wird aus der vorliegenden Systemzeit oder der Alarm-Zeit des Timers mit Hilfe der o.g. anwendungsindividuellen Funktion der Zeitcode errechnet. Dieser dient nun als Schlüssel um aus den Futurekey(s) den/die jeweiligen Ausgangsschlüssel zu dechiffrieren.

6. Aufgabenpuffer

Alle in der Zwischenzeit (also der Zeit von der letzten Beendigung des Timer-Programms bis zu seinem erneuten Aufruf durch den Timer-Event-Interrupt) von anderen Programmen, Servern, Clients, etc. angeforderten Dechiffrierungen und Chiffrierungen von Daten, wurden gepuffert. Dieser Puffer kann eine Art Datenbank, eine einfache Datei oder eine Art Stack/Stapel sein, wie der Prozessor einen führt.
Hierbei ist in jedem Fall sicher zu stellen, dass dieser Puffer nicht manipuliert werden kann. Aufgrund der kurzen Gültigkeit dieses Puffers wäre hierzu z.B. eine asymmetrische Verschlüsselung in Verbindung mit Hash-Prüfwerten und Zertifikaten das geeignete Instrument. Die Ausführungen unter K.3 (Schnittstellen-Kommunikation) sind zu beachten. Optimal wäre das nach G.3 für Passwörter angewandte Verfahren wobei für normale Daten natürlich kein Hashing verwendet werden kann. Es wird letztlich eine Frage der Performance sein. Doch sollte bedacht werden: Es nützt nicht viel wenn die Daten ab dem Eintritt in das vorliegende Verfahren bombensicher sind, aber auf dem Weg dahin jederzeit angegriffen werden können.

Eine gute Alternative wäre die Zwischenspeicherung der Aufgaben auf der Timer-Hardwareerweiterung. Diese kann sich einfach besser vor Manipulationen schützen und sicher stellen, dass wirklich nur das Timer-Programm diese Aufgabenliste erhält.

7. Anmeldedaten werden nie dechiffriert

Die angeforderten und gepufferten Dechiffrierungen und Chiffrierungen werden nun vom Timer-Programm ausgeführt, wobei Dechiffrierungen von Anmeldedaten grundsätzlich unterbunden werden. Dies stellt das Programm sicher indem die unterschiedlichen Spalten einer Datenbank unterschiedlich verschlüsselt werden (arithmetische Bearbeitung des Masterkey) so dass eine Manipulation, wie z.B. ein per SQL-Angriff denkbarer Tausch der Datenbankspalten, ausgeschlossen ist und das Timer-Programm immer erkennen kann wenn es sich um Anmeldedaten handelt, bzw. nach so einem Tausch, die beabsichtigten Dechiffrierungen fehl schlagen, da die arithmetische Veränderung des Masterkey aufgrund Nummer der Datenbankspalte, sowie Daten aus den vorhergehenden und nachfolgenden Spalten, dann nicht zum für diese Spalte passenden Masterkey führen.

Nach Abschluss der aufgelaufenen Dechiffrierungen und Chiffrierungen wird der Ausstieg des Timer-Programms eingeleitet, was die zu Anfangs beschriebenen Verfahrensschritte wie Festlegen einer neuen Zeit X, daraus Zeitcode berechnen und damit die Ausgangsschlüssel zu

Futurekeys zu chiffrieren, Timer setzen sowie Ausgangsschüssel, Zeitcode und Zeit X zu löschen usw. bedeutet. (Pos. 1 bis 3 aus Fig. 1) Das Verfahren wiederholt sich also.
Da die Ausgangsschlüssel nun de facto nicht mehr vorhanden sind und auch bis auf weiteres kein Schlüssel existiert mit dem man die Futurekeys wieder zu Ausgangsschlüsseln dechiffrieren könnte, die Ausgangsschlüssel dann aber doch von außen unvorhersehbar plötzlich (Zeit X) wieder da sind, nennt der Erfinder dieses Verschwinden lassen der Ausgangsschlüssel „ghosten". (siehe Fig. 1)

Die einhergehende Verzögerung der Anmeldung dadurch dass im Durchschnitt erst in ca. 1,5 Sekunden die Anmeldedaten geprüft werden, fällt (vor allem im Internetverkehr) nicht ins Gewicht da sie für jeden Anwender nur einmalig bei Anmeldung auftritt. Werden mit dem Verfahren auch weitere Daten verschlüsselt auf die immer wieder zugegriffen werden muss, so ist die Intervall-Vorgabe ggf. kürzer zu wählen (z.B. 0,2 Sek.).

D. BESCHREIBUNG ANSPRUCH 2: (TIMER-HARDWAREERWEITERUNG)

Verfahren nach Anspruch 1, wobei
der verwendete Timer aus Verfahrensschritt c) bis d) des Anspruch 1 auf einer zusätzlichen Hardware, der Timer-Hardwareerweiterung, untergebracht ist und nach dem Starten des Timers bis zu seinem Ablauf/Auslösen gar nicht und danach nur jeweils einmal gelesen werden kann.

1. Timer (Grundsätzliches)

Ein Timer ist eine software- oder hardware-verwirklichte Einrichtung welche zeitbezogene Funktionen bietet, wie Uhr, Alarmzeit, Zähler, Stoppfunktion und Countdown,
Die Mindestfunktion die in Anspruch 1 benötigt wird ist die sog. Countdown-Funktion. Hier wird eine Zeit angegeben und der Timer zählt rückwärts bis Null erreicht ist. Dann gibt der Timer ein Alarm-Signal, was in der Definition der Ansprüche mit „Ablauf" gemeint ist.
Eine andere Variante ist die Alarm-Funktion. Hierzu sind mindestens 2 Register notwendig. Im Zeit-/Zähl-Register wird eine Zeit, wie z.B. die aktuelle Uhrzeit angegeben. Das Alarm-Signal wird ausgelöst sobald das Zeit-Register die im Alarm-Register eingetragene Zeit erreicht hat.

Dies ist in den Ansprüchen mit „Auslösen“ gemeint.
Beide Varianten sind im Sinne der Erfindung denkbar doch nur die Alarm-Funktion bietet 100%-ige Sicherheit und ist somit diejenige von der im gesamten Dokument in erster Linie auszugehen ist. Die Countdown-Variante wird nur der Vollständigkeitshalber für Systeme mit einbezogen, die die Alarm-Funktion nicht bieten können.

2. Interrupts / NMI

In den meisten Computer-Systemen werden Hardware-Timer als elektronische Timer-Bausteine oder Timer-Komponenten in bestehenden Chips integriert. Das Alarm-Signal wird zumeist mit einem Interrupt-Eingang der CPU verbunden so dass die Alarm-Funktion programmtechnisch zeitnah abgearbeitet werden kann. Erhält eine CPU einen Interrupt, wird die Ausführung der laufenden Programme unterbrochen. Programmzähler und CPU-Register werden im Stack gespeichert und die CPU ruft das Programm auf, das in der Interrupt-Sprung-Tabelle eingetragen ist. Dabei wird zunächst noch festgestellt um welchen Interrupt es sich handelt und das dazu passende Programm gestartet. Die meisten Systeme haben 16 unterscheidbare Interrupts. Zusätzlich bieten alle dem Erfinder bekannten Systeme einen besonderen Interrupt. Den Not Maskabel Interrupt (NMI). Dieser kann im Gegensatz zu den normalen Interrupts nicht verhindert werden. Er hat die höchste Priorität. Am Ende des Interrupt-Programms werden die CPU-Register wieder vom Stack geholt und das ursprünglich laufende Programm fortgesetzt. Daraus ergeben sich für Interrupt-Programme einige besondere Anforderungen die jedoch hier nichts zur Sache tun.
Entscheidend ist, dass der Alarm-Ausgang des Timers mit dem NMI (oder notfalls Interrupt) des Systems verbunden ist und der NMI-Sprung-Verweis bzw. der entsprechende Eintrag in der Interrupt-Sprung-Tabelle auf das Timer-Programm zeigt und von verschiedener Seite kontrolliert und sichergestellt wird dass dieser Eintrag nicht manipuliert wird. (siehe dazu insbesondere N. Anspruch 9)

3. Register-Größe und Zeit-Auflösung

Timer gibt es in verschiedensten Ausführungen. PC-Timer haben heutzutage i.d.R. eine Größe von 32 bis 64 Bit. D.h. die Zeit, Alarm-, Countdown-Zeit sind jeweils 32- oder 64-Bit-Register. Oftmals findet sich dabei nachfolgende Aufteilung:

hhmmssHH Wobei jeder Buchstabe für ein Byte steht. 2 Byte für Stunden: hh, 2 Byte für Minuten: mm, 2 Byte für Sekunden: ss, 2 Byte für 1/100stel-Sekunden: HH. 8 Bytes entsprechen

64 Bits. Manchmal gibt es noch ein weiteres 64-Bit-Register für das Datum im Format YYYYMMDD.

Es gibt aber auch viele Timer die einfach als ein in Milli- oder Mikrosekunden getakteter Zähler aufgebaut sind. Ein 32-Bit Register könnte demnach 2^32 = 4,2e9 Millisekunden zählen. So könnten bis 1193 Stunden gezählt werden was also locker für einen Tag reicht. Für eine Auflösung in Mikrosekunden sind wenigstens 11 Stellen notwendig, was mindestens 37 Bit benötigt. Es gibt zwar einige Exoten mit 48-Bit aber die meisten gehen dann schon auf 64-Bit. Auch das Datum wird dann einfach als Zahl geführt, wobei unser Kalender noch lange mit einer 6-stelligen Zahl an Tagen auskommt. So könnte man ein 64-Bit-Register in 24 Bit für Tage und 40 Bit für Mikrosekunden aufteilen und käme bis ins Jahr 45964 klar.

Wir benötigen für ein wirklich sicheres Verfahren zumindest ein 128-Bit-Register bzw. 2 davon (eines für die laufende Zeit und eines für die Alarmzeit). 64-Bit funktioniert auch noch, hält aber bestimmten sehr unwahrscheinlichen Angriffszenarien der Zukunft nicht stand. (siehe Angriffszenarien, Geringe Timer-Auflösung und Schlüsselbreite + Technischer Fortschritt) Wünschenswert ist außerdem eine höhere Zeitauflösung als Mikrosekunden. Ein Taktgeber bei 1 GHz zählt bereits Pikosekunden. Schnellere sind dem Erfinder nicht bekannt, aber das kann ja noch werden. Der Aufbau der Timer-Register spielt indes keine Rolle. Wichtig ist nur, dass sie beide beliebig gesetzt werden können. D.h. für ein sicheres Verfahren nutzen wir einen Alarmzeit-Timer. Ein Countdown-Timer müsste eine Auflösung (kleinste Zeiteinheit) von 1e-40 Sekunden bieten und ist somit völlig unrealistisch.

4. Timer-Hardware-Erweiterung

Die Standard-Timer in PC-Systemen bieten zu 99% die o.g. Zeitfunktion und eine oder mehrere Alarmfunktion(en), doch lassen sich damit einige der für die Erfindung relevanten Merkmale nicht umsetzen. Zudem bieten sie oft nur eine Auflösung von 1 Millisekunde.

Aus diesem Grund ist ein zusätzlicher Hardware-Timer Teil der Erfindung. Dieser wird zur Realisierung der höchsten Sicherheitsstufe des vorliegenden Datenschutzsystems benötigt. Er kann auf einer Einschub-/Erweiterungskarte untergebracht werden wie sie auch für andere Hardware-Erweiterung von klassischen PC-Systemen üblich sind und über ein Bus-System mit der Hauptplatine verbunden werden. Üblich sind heute sog. PCI-Karten, wobei eine Verbindung – wenn möglich – mit dem internen Bus zu bevorzugen ist. Denkbar wäre auch eine Art Huckepack-System bei dem ein bestehender Chip ersetzt wird durch eine Erweiterung die diesen ersetzten Chip simuliert aber gleichzeitig weitere Komponenten mit dem internen Bus verbindet.

Zukünftige Systeme könnten ihn auch bereits auf der Hauptplatine haben, oder er ist in einem anderen Chip mit untergebracht.

Für die Erfindung ist nur von Belang dass die nachfolgend dargestellten Dienste zur Verfügung gestellt werden. Es wäre auch denkbar den 128-Bit-Timer durch die Verkettung von 2 64-Bit-Timern zu realisieren.

5. Lesekontrolle

Wichtigste Eigenschaft des Timers ist, dass er nicht ausgelesen werden kann bis er abgelaufen ist (Countdown bei 0) bzw. Alarm ausgelöst (Alarmzeit erreicht) hat. Dies gilt für das Alarmzeitregister genauso wie für das Zeitregister (bzw. bei einem Countdown-Timer für das Countdown/-Zählregister).

a) Lese-Sperre bis Alarm

Dies muss hardwareseitig sichergestellt sein so dass eine (vor allem softwareseitige) Manipulation / Veränderung ausgeschlossen ist. Möglich wäre dies im einfachsten Sinne durch eine Auskopplung des Read-Eingangs des Timers so lange das Alarm-Signal auf inaktiv steht. Das Alarm-Signal koppelt das Read-Signal wieder ein so dass dies den Timer erreichen kann. Doch bereits mit dem ersten Eintreffen eines Read-Signals wird die Auskopplung wieder aktiv so dass weitere Lesevorgänge erst wieder nach erneuter Alarm-Auslösung möglich sind, was das nachfolgende Leistungsmerkmal verwirklicht. (Fig. 7)

Im weiteren Verlauf dieser Beschreibung werden noch weitere Leistungsmerkmale einer potentiellen (Timer-)Hardwareerweiterung vorgestellt. Es wird daher Sinn machen diese Hardware mit einem Mikrocontroller aufzubauen womit der Signalsteuerung alle Möglichkeiten offen stehen, so wie eben auch der Kontrolle der Lesbarkeit.

b) Einmalige Info / Nachlaufen

Zweitwichtigstes Merkmal ist sicher zu stellen, dass nach dem Alarm-Ereignis der Timer (sein Alarm-Register) einmal ausgelesen werden kann. Dies ist notwendig um dem Timer-Programm die Zeit X (die gesetzte Alarmzeit mit welcher das Timer-Programm den Zeitcode wieder herstellen kann) oder die Verzögerung seit Zeit X zur Verfügung zu stellen. Dies darf aber nur einmal möglich sein, damit kein anderes Programm (als das Timer-Programm) bzw. keine andere Hardware diese sensible Information abrufen kann.
Im Fall eines Countdown-Timers muss dieser nach dem Alarm-Ereignis (Timer hat 0 erreicht)

weiter laufen(quasi ins Minus) bis er gelesen wird.
Dies ist Teil der Fehlerkorrektur zu Anfangs des Timer-Programms (siehe Kapitel J).

c) Manipulations-Alarm bei unberechtigtem Lesen

Zusätzlich sollte der Timer einen System-Alarm auslösen können wenn vor dem Timer-Alarm-Ereignis versucht wird, den Timer auszulesen, oder versucht wird den Timer ein zweites mal auszulesen. System-Alarm könnte über einen weiteren Interrupt mit einer anderen ID oder über ein Stopp-Signal realisiert werden. Auch die Verbindung zu einem Tongeber bzw. ein solcher auf der Hardwareerweiterung wäre hilfreich. Hardwareseitig ließe sich das mit einer einfachen UND-Logik zwischen „Read ausgekoppelt“ und „Read-Eingang“ realisieren.
Somit würde sofort erkannt werden, dass ein Schadprogramm im System sein Unwesen treibt.

Es ist fraglich ob es von Vorteil wäre wenn zusätzlich das Schreiben des Timers nur zulässig sein sollte, nachdem dieser abgelaufen ist. Dies wäre zumindest für das allererste Setzen u.U. problematisch, doch vor allem kann mit einem illegalen Schreiben der Datenschutz keinesfalls ausgehebelt werden. Wird der Timer manipuliert so gibt er zu einer anderen Zeit als Zeit X Alarm. Das Ergebnis wäre dass das Timer-Programm nicht den korrekten Zeitcode ermittelt und die Entschlüsselung der Futurekeys versagt bzw. falsche Ergebnisse bringt. Als Anzeichen eines unberechtigten Manipulationsversuchs sollte es dennoch ausgewertet werden.

6. Übertragungsverschlüsselung

Somit ist der Timer nur ein einziges Mal pro Phase (Start – Ablauf/Alarm) auslesbar. Im Falle eines Countdown-Timers läuft dieser nach dem Countdown weiter bis er ausgelesen wurde. Die gelesene Nachlaufzeit in Verbindung mit der Systemzeit oder die Alarmzeit ist somit nur dem Timer und dem Timer-Programm bekannt und kann genutzt werden um damit die neu zu setzende Timerzeit arithmetisch zu verändern und/oder zu verschlüsseln. Dadurch kann diese neue Zeit X nicht anderweitig abgefangen werden bzw. wäre dann nutzlos.
Auf einer Mikrocontroller-Karte als Timer-Hardwareerweiterung wäre eine Verschlüsselung mit einem entsprechenden Chip oder programmtechnisch zu bewerkstelligen. Ohne Mikrocontroller wäre dann wohl eine arithmetische Bearbeitung die sinnvollste Lösung. Da nur etwa 3 Sekunden bis zur nächsten Timer-Phase bleiben, dürfte auch eine hochwertige arithmetische Bearbeitung wohl kaum zu knacken sein, zumal bei der Alarmzeit -Variante ohnehin nur Pseudo-Zufalls-Zeiten als Basis genommen werden.
Die Art der arithmetischen Bearbeitung kann dabei in den ersten Bits oder Bytes der Alarmzeit, der Timer-Zeit oder der Systemzeit abzüglich Nachlaufzeit codiert sein, sodass nicht nur die

Parameter einer arithmetischen Bearbeitung, sondern auch die Rechenoperationen bzw. -arten variieren.

Als zusätzliche Sicherheitsmaßnahme kann die Timer-Hardwareerweiterung System-Alarm auslösen wenn sie nicht binnen einer bestimmten Zeit (in etwa die Zeit die dem Timer-Programm für die Abarbeitung der aufgelaufenen Aufgaben gegeben wird) nach dem Auslösen/Alarm erneut gestartet wird. Dies würde darauf hinweisen, dass ein ernsthaftes Problem vorliegt.

E. BESCHREIBUNG ANSPRUCH 3: (RAM-MATRIX – VERSTECK AUSGANGSSCHLÜSSEL)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
jegliche kryptografische Schlüssel, insbesondere im Verfahrensschritt c) des Anspruch 1 wieder errechnete Ausgangsschlüssel, wenn Sie im Arbeitsspeicher, oder einem anderen Speicher, gespeichert werden, wie dies während deren Verwendung für Dechiffrierungs- und Chiffrierungsaufgaben des Timer-Programms der Fall sein könnte, entweder in mehreren Teilen aufgesplittet in einem Speicherbereich aus Zufallszahlen, der Zufallsmatrix, versteckt gehalten werden, wobei die Zeiger darauf ausschließlich in Prozessorregistern abgelegt werden oder die Schlüssel selbst werden ausschließlich in Prozessorregistern gespeichert; wobei der Speicherbereich für die Zufallsmatrix jedes Mal bevor darin Schlüssel versteckt werden mit neuen Zufallszahlen zu füllen ist.

1. Problem Multithreading

Moderne Systeme sind so aufgebaut dass mehrere Threads (Programme, Programmteile, Aufgaben, Dienste etc.) gleichzeitig ablaufen. Oft bedeutet(e) „gleichzeitig“ nur, dass der Prozessor seine Ressourcen aufteilt und auf die unterschiedlichen Threads je nach deren Priorität verteilt. Im Zeitalter der Multi-Core-CPU’s laufen Threads jedoch tatsächlich gleichzeitig ab, zumindest soweit das möglich ist, denn der Zugriff auf RAM usw. ist natürlich immer nur von einem Thread gleichzeitig möglich, weswegen der Prozessor-Cache in dieser Hinsicht wichtig ist.

Zwar wird von einem Großteil moderner Betriebssysteme ein Übergriff des Arbeitsspeichers zwischen unterschiedlichen Threads unterbunden, doch es ist immer vorstellbar dass ein Thread Wege findet derartige Begrenzungen zu umgehen. Es ist daher nicht undenkbar dass ein bösartiger Thread in der Lage ist, auf den Speicherbereich den das Timer-Programm benutzt und in dem Daten wie auch die entschlüsselten Futurekeys (die Ausgangsschlüssel) liegen, zuzugreifen und diese somit zu erbeuten. Dies muss natürlich unbedingt vereitelt werden.

Eigentlich sollte das System so aufgebaut werden, dass es möglich ist, das Timer-Programm völlig exklusiv ablaufen zu lassen, was bedeutet, es gibt im gesamten System keine anderen Programme die parallel zum Timer-Programm abgearbeitet werden. Es gibt Systeme bei denen ein Interrupt, insbesondere ein NMI, ohnehin exklusiv läuft. Falls dem nicht so ist, muss das Timer-Programm an dessen Anfang alle anderen Threads des Systems deaktivieren, so wie auch alle anderen Interrupts. Das System sollte entsprechend angepasst werden. Für den Fall dass dies nicht möglich ist oder der Angreifer einen Weg findet diese Beschränkungen zu umgehen, nutzt die vorliegende Erfindung eine

2. Speicher-ZufallsMatrix

Hierzu werden die zu schützenden Schlüssel, wie z.B. die Ausgangsschlüssel (wir gehen im Wesentlich von 2 aus, Masterkey/Zeitschlüssel, siehe auch weiter unten) in mehrere (mind. 8 besser 16) Teilen aufgesplittet, an stets unterschiedlichen Stellen innerhalb einer Speicher-ZufallsMatrix abgelegt. Hierzu wird am Anfang des Timer-Programms ein größerer (z.B. 16 MB in einem 64-Bit-System) geschützter Speicherbereich mit (z.B. 1 Million (unterschiedlichen) 128-Bit-) Zufallszahlen gefüllt und per Zufall bestimmt wo in diesem Bereich genau der Schlüssel abgelegt wird, wobei er nie als ganzes sondern in vorzugsweise 16 Teile aufgesplittet gespeichert wird und die Position jedes dieser Teile (in bestimmten Grenzen) per Zufallszahl bestimmt wird.

Wird der in Punkt G. 1. (Fehlerkorrektur, bei Verwendung eines Countdown-Timers) dargestellte Prüfziffern-Hash verwendet, müssen die Zufallszahlen so gewählt werden, dass 80% aller Kombinationen von Zufallswerten die die selbe Bitlänge des zu versteckenden Schlüssels ergeben (also z.B. 128 Bit) zumindest einmal, besser mehrfach (je nach Reihenfolge) die zur Fehlerkorrektur vermerkte 16-Bit-Hash-Prüfziffer ergeben.

Beispiel: Ein 128-Bit-Schlüssel wird in 16 Teilen in einem Speicherbereich von 16 MB (Megabyte) versteckt. Der angenommene 16-Bit-Prüfziffern-Hash des Schlüssel beträgt $A7. So wird zunächst eine Menge von Byte-Zahlen (8 Bit = 128 Bit / 16 Teile) gebildet die in 80% aller

möglichen 128-Bit-Kombinationen (egal in welcher Reihenfolge man sie zu einer 128-Bit-Zahl zusammen setzt) den Prüfziffern-Hash $A7 ergeben. Nun wird etwa ¼ des Speicherbereichs zufällig mit 8-Bit-Werten dieser Menge gefüllt. Anschließend werden per Zufall 128-Bit-Werte erzeugt und so lange verworfen bis sie den selben 16-Bit-Prüfziffern-Hash des Schlüssel ($A7) ergeben. Nur solche werden dann in 16 Teilen gesplittet per Zufall in den Speicherbereich geschrieben.
Danach wird der tatsächliche Schlüssel (Ausgangsschlüssel) ebenso in den Speicherbereich geschrieben.
Sollten 2 Schlüssel versteckt werden müssen, ist es insofern einfacher, als dass die zufällig erzeugten Werte jeweils zwei Prüfziffer-Hash-Werte ergeben können. Die Untermenge verdoppelt sich somit.

Bei der hier beispielhaft dargestellten Matrix ergäben sich im Falle eines Brute-Force-Angriffs etwa 1e115 verschiedene Kombinationen für einen 128-Bit-Schlüssel. Dies entspricht 2^382 Möglichkeiten. Das reicht natürlich für die kurze Zeit in der diese Matrix existiert und auch noch für den Fall dass ein Angreifer die Matrix komplett erbeuten und dann in „aller Ruhe“ alle Möglichkeiten durch rechnen könnte. Selbst wenn der Inhalt des Arbeitsspeicher samt Matrix UND die gesamte Datenbank erbeutet wird reicht die Sicherheit noch locker für die nächsten 300 Jahre (siehe technischer Fortschritt). Naja, und 2 versteckte Schlüssel bringen ohnehin auch schon 2e230 Möglichkeiten. Da würde dann einfach das direkte Brute-Forcing von 2 128-Bit-Schlüsseln mit insgesamt mit 1e77 Möglichkeiten mehr Sinn machen. Das bedeutet auch: Bei 2 64-Bit-Zeigern ist eine 3-MB-SpeicherMatrix ausreichend. Zeiger-Größe und Matrix-Größe sind direkt proportional zueinander.

Wie schon zu Anfangs dieser Beschreibung dargestellt nützt das beste Versteck jedoch nichts wenn jemand den Versteck-Hinweis, sprich den Zeiger dahin finden kann.
Daher werden die Zeiger welche auf z.B. 2 so versteckte Schlüssel bzw. deren z.B. 16 Teile zeigen mit Hilfe eines anwendungsindividuellen Algorithmus in zwei (je einem) QWords (64-Bit) codiert. Bei einer Aufsplittung in je 8 Teile reicht auch ein einziges QWord. Dieser Algorithmus enthält mehrere Zufallswerte wodurch eine Errechnung der jeweiligen Position so schwer ist, dass zumindest der dann erbeutete Zeitschlüssel (siehe später) längst ungültig sein dürfte. Es muss jedoch auch klar sein: Zwei 64-Bit-Zeiger bergen 2^128 Bit Möglichkeiten. Wenn also jemand den anwendungsindividuellen Algorithmus kennt, mit dem die Zeiger codiert werden, reduzieren sich die o.g. Möglichkeiten von 1e77 auf 3e38. Hat also jemand tatsächlich Matrix UND die gesamte Datenbank erbeutet und die Ressourcen einen Angriff mit z.B.

beachtlichen 10 PetaFlops (10t-schnellster SuperComputer der Welt) zu rechnen, dann dauert es heute womöglich noch Billionen Jahre aber in 50 Jahren könnte er in einer Woche durch sein. Ergo: Bei Daten, die länger als 50 Jahre geschützt bleiben sollen, muss eine Rechner- bzw. Prozessor-Architektur gewählt werden die auch 2 128-Bit-Zeiger im Prozessorregister unterbringen kann. Ganz nebenbei erübrigt sich dann der ganze Aufwand mit der Matrix da man dann die zu schützenden Schlüssel (so diese nicht länger als 128 Bit sind) auch direkt in den Prozessorregistern ablegen könnte. Warum sind diese Prozessorregister eigentlich so entscheidend? Das liegt daran, dass man darauf, auch aus anderen Threads / Prozessorkernen heraus nicht zugreifen kann. Die Register würden nur im Falle eines Interrupts (was das Timer-Programm ja selbst ist und insofern hier keine Rolle spielt) im Stack gespeichert aber ansonsten gibt es für die Außenwelt nur im Falle eines sog. Prozessor-Dumps die Möglichkeit da ran zu kommen. Bei einem Dump sichert der Computer den kompletten Inhalt des Prozessors und meist bestimmter Bereiche des Arbeitsspeichers auf Festplatte und hält das System an oder macht einen Reset / Neustart. Dies geschieht zumeist bei einem schwerwiegenden Fehler der es der CPU unmöglich macht, das System weiter auszuführen. Da dies ein ernstzunehmendes Angriffsszenario ist, muss die Möglichkeit eines solchen Prozessor-Dumps unbedingt deaktiviert werden.

Abgesehen von den Prozessorregistern ist nur noch der Prozessorstack als kurzzeitiger Zwischenspeicherort für einen der QWord-Zeiger zulässig, wobei er dann vorher mit dem jeweils anderen Schlüssel (den Schlüssel auf den das andere QWord zeigt) oder mit Hilfe der Timer-Hardwareerweiterung verschlüsselt werden sollte. Besser wäre jedoch aus Performance-Gründen, diesen Zeiger zumindest mit dem anderen Zeiger (der im Prozessorregister verbleibt) oder dem damit verbundenen Schlüssel arithmetisch zu bearbeiten, was nicht so aufwendig wie eine Verschlüsselung ist aber für die kurze Zeit eine vergleichsweise hohe und akzeptable Sicherheit birgt.

Bei entsprechender Ausgestaltung der Timer-Hardwareerweiterung kann ein Zeiger auch auf dieser zwischengespeichert werden.

Der Speicherbereich für die Speicher-ZufallsMatrix kann gerne großzügig gewählt werden. 16 MB wäre jedoch mehr als ausreichend. Die Größe ist im Wesentlichen nur dadurch begrenzt dass man in der Lage sein muss, die Zeiger dafür ausschließlich in Prozessorregistern abzulegen. Dadurch können diese aufgrund der begrenzten Anzahl und Größe der Register in vielen Systemen nicht sehr groß sein. Es sollte aber immer der größtmögliche Speicher für die Zufallsmatrix gewählt werden. D.h. stehen genug Register zur Verfügung die zum Ablegen der

Zeiger genutzt werden können, dann sollten diese Möglichkeiten auch genutzt werden und z.B. 16 MB für die Zufallsmatrix angelegt werden.
Jedoch muss auch noch die Performance bedacht werden, denn es bedarf natürlich mehr Zeit, 128 MB mit Zufallszahlen zu füllen als nur 16 MB. Und es ist zu beachten, dass dieses Füllen der Matrix mit Zufallszahlen bei jedem Timer-Programm-Durchgang, also vor jeder Speicherung von Ausgangsschlüsseln darin (etwa alle 3 Sek.), neu gemacht werden muss. Ansonsten bräuchte ein Angreifer nur die eine Matrix mit der anderen vergleichen und könnte anhand der Veränderungen erkennen wo die Schlüssel versteckt wurden.
Unterm Strich bringt eine Vergrößerung der Matrix nur dann etwas wenn sich daraus deutlich mehr Möglichkeiten ergeben wie aus der Decodierung der Zeiger.

Alternativ kann statt der Speicherung von Masterkey und Zeitschlüssel auch nur der Zeitcode in der Matrix versteckt werden, wenn die Rechenpower es zulässt dass für jede Entschlüsselung von Daten zunächst aus den Futurekeys mit Hilfe des Zeitcodes die jeweiligen Ausgangsschlüssel (Masterkey und Zeitschlüssel) errechnet werden unmittelbar bevor diese zur Entschlüsselung verwendet werden. Aus Performance-Gründen kommt diese Ausgestaltung jedoch kaum in Betracht.

Eine sicherheitssteigernde Ergänzung ist zu realisieren indem der für die Matrix verwendete Speicher sehr langsam ist und somit eine komplette Erbeutung innerhalb der Gültigkeit der dort versteckten Schlüssel schon hardwaretechnisch ausgeschlossen ist. (Siehe O.7. Flash-/RAM-Speicher)

F. BESCHREIBUNG ANSPRUCH 4: (DATENSCHUTZ-SYSTEM)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
zum Schutz von Daten diese nacheinander mit mindestens 2 unterschiedlichen Ausgangsschlüsseln mit jeweils unterschiedlichen Verschlüsselungsverfahren verschlüsselt werden, wobei
a) einer dieser Ausgangsschlüsseln, der Masterkey, jedes Mal bevor er zur Verschlüsselung von Daten eingesetzt wird, so bearbeitet wird, dass dies für die Entschlüsselung wieder rekonstruiert werden kann und dennoch möglichst jede Verschlüsselung mit einem etwas anderen Masterkey erfolgt und

b) ein anderer Ausgangsschlüssel, der Zeitschlüssel, in bestimmten Zeitabständen immer wieder neu erzeugt wird, wobei dann jeweils alle damit verschlüsselten Daten mit dem bisherigen Zeitschlüssel entschlüsselt und sofort mit dem neu erzeugten Zeitschlüssel wieder verschlüsselt werden, sodass rotierende Geheimtexte entstehen. (Fig. 3)

Unter Ausgangsschlüssel werden auch hier – analog zu den vorherigen Ansprüchen – jegliche kryptografische Schlüssel verstanden, die zur Verschlüsselung (zum Schutz) von Daten eingesetzt werden können. Der Schutz von Daten umfasst also hier deren mehrfache Verschlüsselung um sie vor unbefugter Einsichtnahme zu schützen. Wenn hier und im Folgenden von Daten die Rede ist, so sind dies jegliche Art von Daten wie sie zumeist in Datenbanken (Terminologie siehe Fig. 2) verwaltet werden. Insbesondere auch Anmeldedaten, also Anmeldenamen und Passwörter. Deshalb wird die Verfahrensweise auch anhand einer zu schützenden Datenbank dargestellt.
Das Verfahren kann grundsätzlich mit einer unbegrenzten Anzahl Schlüssel angewendet werden, wir gehen hier und im weiteren Verlauf dieser Beschreibung jedoch stets von 2 Schlüsseln aus, wobei der eine Masterkey heißt und der andere Zeitschlüssel. (Siehe Fig. 3)
Sie sollten jeweils eine Länge von mindestens 128 Bit haben. Empfohlen wird jedoch – vor allem bei Daten die auch in 50 Jahren noch sicher sein sollen, zumindest 192-Bit-Schlüssel und bei besonders sensiblen Daten 256-Bit-Schlüssel oder längere einzusetzen. Da aufgrund des Aufbau des kompletten Verfahrens bei einem Angriff letztlich wohl nur die Brute-Force-Methode zum Einsatz kommen dürfte, gilt hier allerhöchstwahrscheinlich: Je länger desto sicherer.

In der vorliegend angenommenen Ausgestaltung werden zu schützenden Daten bzw. gehashte Anmeldedaten zunächst mit einem je nach Datenspalte oder Datenfeld veränderten Masterkey und anschließend mit einem zeitlich wechselnden Zeitschlüssel verschlüsselt.

1. Masterkey

Der Masterkey ist als anwendungsindividueller und nahezu einmaliger Schlüssel vorgesehen. Er wird, bevor er zur Verschlüsselung von Daten eingesetzt wird, - soweit möglich - (arithmetisch) bearbeitet bzw. verändert. Diese Bearbeitung erfolgt mit einer anwendungsindividuellen Formel auf Basis von Organisationsdaten und/oder sonstigen Daten des jeweiligen Datenbanksatzes. (Pos. 5, Fig. 3)

a) Arithmetische Bearbeitung

Grundlagen dieser Formel können z.B. der so genannte Unique-Key (einmalige Ordnungszahl / Nummerierung jedes Datensatzes) und/oder eine Datenfeldnummer und/oder weitere Daten die je nach zu chiffrierenden Datenfeld wechseln, und/oder weitere Daten des jeweiligen Datenbanksatzes (z.B. Anmeldename, Klarname oder Geburtsdatum) bzw. ein stark reduzierter (Prüfsummen-) Hashwert davon, sein. Ungünstig sind die jeweiligen Geheimtexte, da die Verschlüsselungen dazu wechseln können (siehe Zeitschlüssel). Gut geeignet ist z.B. der Klartext des Anmeldenamens oder dessen Masterkey-verschlüsselter Geheimtext, der als wichtigstes Such-Datenfeld mit dem unbearbeiteten Masterkey, oder nur spaltensensitiv bearbeitet, verschlüsselt wird da ansonsten eine Suchanfrage an die Datenbank unmöglich oder zumindest sehr zeitaufwendig wäre. Zusätzlich sollte für jede zu verschlüsselnde Datenspalte eine etwas andere Formel zur Bearbeitung des Masterkey angewendet und die Daten jeweils anderer Datenfelder desselben Datensatzes einbezogen werden, so dass Manipulationen durch Tausch der Datenfelder bzw. -spalten zu keiner gültigen Dechiffrierung führen.

Entscheidend dabei ist letztlich, dass die jeweiligen aus der Datenbank bezogenen Parameter für die Bearbeitung des Masterkey wieder genauso zur Verfügung stehen, wenn die jeweiligen Daten wieder entschlüsselt werden sollen. Es muss also darauf geachtet werden dass keine Paradoxen entsteht.

Somit erfolgt nahezu jede Verschlüsselung welche mit dem Masterkey ausgeführt wird mit einem etwas anderen Masterkey, wobei es nicht notwendig ist, dies gesondert zu kontrollieren oder sicherzustellen. Jedoch ist die Formel so zu wählen, dass ein Mehrfach-Vorkommen desselben Masterkey eher unwahrscheinlich ist.
Ein Beispiel für eine solche Formel findet sich unter Punkt I 3.

Soll nach Daten in einer verschlüsselten Datenbank gesucht werden, so wird der zu suchende Wert gleichfalls verschlüsselt und das Ergebnis dann mit den bereits verschlüsselten Werten in der Datenbank verglichen bzw. aussortiert (bzw. die Hashwerte davon). Aus diesem Grund funktioniert eine Datenbank-Suche nicht, wenn die Daten z.B. mit jedem Datensatz anders verschlüsselt werden. „Soweit möglich“ (Absatz 1 unter 1. Masterkey), bzw. „möglichst“ in der Anspruchsdefinition, bezieht sich genau darauf, da die arithmetische Bearbeitung des Masterkeys für Datenfelder bzw. Datenspalten, in denen eine Datenbank-Suche möglich sein soll, nur eingeschränkt möglich ist. Eine spaltensensitive Bearbeitung bleibt möglich aber innerhalb der Spalte muss der Masterkey immer gleich bearbeitet sein bzw. der stets gleiche Masterkey’ eingesetzt werden. Eine Bearbeitung des Masterkeys, die den jeweiligen Masterkey’

mit jedem Datensatz verändert kann nur durchgeführt werden für Datenspalten zu denen einen Datenbank-Suche verzichtbar ist.

b) Entstehung des Masterkeys

Dieser Masterkey muss im Gegensatz zu dem verwendeten Algorithmus geheim und insofern auch individuell zu der jeweiligen Anwendung sein. Er kann z.B. beim Installationsvorgang (der Software für dieses Verfahren) aus einer Programm-Seriennummer oder aus den Hardware-Daten des PC's (inkl. div. Seriennummern und Prozessor-IDs) oder aus einer z.B. 58-stelligen Zufallszahl evtl. in Kombination mit einem Unique-Key der in den Programmcode implementiert wurde generiert werden. Es wäre denkbar, dass sich die Anwendung oder der Anwender diesen bei der Registrierung bzw. nach der Installation vom Hersteller bzw. dessen Registrierungsserver zuweisen lässt. Auch eine Generierung aus einem Hardwarebaustein (Hardwareerweiterung lt. Anspruch 2, 8, 9) zur Erzeugung von Schlüsseln/Zufallszahlen oder mit Hilfe von zufälligen Mausbewegungen des Anwenders wäre praktikabel. Letztlich wird es wohl eine Kombination aus diesen Methoden sein.

In der Praxis wird es wohl pro System wenigstens zwei Masterkeys geben, wobei einer für die Datenfelder verwendet wird, die aus Gründen der Datensuche / Suchbarkeit nicht mit veränderten Masterkey verschlüsselt werden und einer für alle anderen.

c) Speicherung Masterkey

Der Masterkey wird direkt nach seiner Generierung mit Hilfe der selben Funktion, wie sie das Timer-Programm für Ausgangsschlüssel (wie der Masterkey einer ist) auf Dauer benutzt, in einen Futurekey umgewandelt, wobei die Zeitspanne bis Zeit X dies eine mal nach der Installation deutlich länger ist als dann im laufenden Betrieb. Ebenso wird dann der Masterkey sicher gelöscht und der Timer das erste mal gesetzt.
Das bedeutet, der Masterkey ist selbst nicht mehr existent. Nur noch in verschlüsselter Form als Futurekey wobei selbst der dafür verwendete Schlüssel (Zeitcode) ebenfalls gelöscht ist. Der Masterkey ist „ghosted“ (siehe Kapitel C).
Das bedeutet jedoch andererseits, er muss gesondert für den Fall eines Hardware-Ausfalls/System-Absturz gesichert werden. Hierzu sehen Sie bitte Punkt T. Backup.

2. Zeitschlüssel

Nach der Verschlüsselung mit dem (bearbeiteten) Masterkey erfolgt eine zweite Verschlüsselung mit dem Zeitschlüssel.

a) Verschlüsselungsverfahren

Das hierbei anzuwendende Verschlüsselungsverfahren muss sich möglichst deutlich von dem Verfahren, welches mit dem Masterkey angewandt wurde, unterscheiden. So würde sich beispielsweise für den Masterkey aus heutiger Sicht eine AES-Verschlüsselung anbieten; also z.B. AES-192 mit höchstmöglicher Rundenzahl, mindestens 12, besser 16 und für den Zeitschlüssel z.B. Twofish oder Blowfish.

Das vorliegende Schutzverfahren ist aber grundsätzlich von den darin verwendeten kryptologischen Verschlüsselungsverfahren selbst unabhängig. Diese sollte mit fortschreitender Zeit und Technik angepasst werden so dass hier jeweils solche Verwendung finden, die zu der jeweiligen Zeit als sicher (und zukunftssicher) bzw. als die Sichersten gelten. Selbiges gilt für die Schlüsselbreite. Je nach Sensibilität der Daten kann die Mindestgröße beliebig von 128 Bit z.B. auf 192, 256, 512, 1024 Bit oder weiter erhöht werden. Letztlich ist es nur eine Frage der Rechenpower.

b) Zeitspanne

Der Zeitschlüssel ist der Schlüssel der in weitgehend regelmäßigen Abständen vom Datenschutz-Programm, hier das Timer-Programm, erneuert wird.
Dafür enthält dieser Schlüssel einen versteckten groben Hinweis darauf, wann er erstellt wurde. Z.B. Stunde und 10er-Minute, was in einem Byte unterzubringen ist. So kann das Timer-Programm bei jedem Durchlauf selbst entscheiden wann es eine Erneuerung einleitet.
Ähnlich wie bei den in gewissen Rahmen variablen Zeitabständen für Zeit X wird auch hier eine grobe Zeitspanne vom Anwender vorgegeben und diese dann um eine zufällige Komponente erweitert.

Angenommen diese liegt bei 90 Minuten, bei einer Varianz von +- 20 Minuten. Das Timer-Programm, welches beispielsweise alle 3 Sekunden ausgeführt wird, erzeugt ab dem Moment ab dem der Zeitschlüssel 70 Minuten alt ist, eine Zufallszahl von 0 – 600 (40 x 20). Sollte 1 als Ergebnis kommen wird die Erneuerung des Zeitschlüssel eingeleitet. Statistisch gesehen sollte dies einmal in diesen 600 Durchläufen (=40 Min.) geschehen. Da dies natürlich nicht verlässlich ist, wird das Timer-Programm die Zahlenmenge mit zunehmendem Fortschritt, z.B. um 10 je Minute die der Zeitschlüssel älter als 70 Minuten ist, verringern. So würde ab 90 Minuten Alter bereits nur noch eine Zufallszahl von 0 – 400, ab 120 Minuten Alter von 0 – 100 und ab 129 Minuten Alter von 0 – 10 erzeugt. Da die Untergrenze bei 0 – 2 liegt, nimmt somit die Wahrscheinlichkeit für eine 1 mit zunehmendem Alter des Zeitschlüssels exponentiell zu.

Wie groß die vom Anwender vorgegebene Zeitspanne (im obigen Beispiel 90 Min. +- 20) ist, hängt im Wesentlichen von der Größe der geschützten Datenbank (bzw. der Menge der geschützten Daten) in Relation zur Rechenleistung ab. Ist diese Relation sehr ungünstig kann die grobe Zeitspanne auf bis zu 24 Stunden +- 5 Stunden gesetzt werden. Darüber ist nicht empfehlenswert. Die Varianz sollte im Bereich von 15 bis 30% der groben Zeitspanne liegen.

c) Erneuerung

Zur Erneuerung des Zeitschlüssel wird zunächst ein neuer Zeitschlüssel' per nicht-deterministischer kryptografisch sicherer Zufallszahl erzeugt. (Siehe Punkt K.1 Zufallszahlen) Diese muss unvorhersehbar und unbeeinflussbar sein. Es versteht sich von selbst dass ggf. durch Mehrfacherzeugung aus einer kleineren Zufallszahl ein z.B. 128-Bit-Schlüssel zusammen gebaut wird.

Nun werden alle mit dem bisherigen Zeitschlüssel verschlüsselten Geheimtexte damit Stück für Stück entschlüsselt und sofort mit dem neuen Zeitschlüssel' verschlüsselt bis alle Daten mit dem Zeitschlüssel' verschlüsselt sind. Dann wird der alte Zeitschlüssel sicher gelöscht und es gilt ab sofort nur noch der neue Zeitschlüssel' als gültiger Zeitschlüssel.

Es ist durchaus denkbar dass aus Performance-Gründen dies auch in Etappen erfolgen könnte, wobei bei Anwendung an einer Datenbank, damit keine Inkonsistenz auftritt, dann zunächst die neuen Geheimtexte aus Zeitschlüssel' in neue Datenbankspalten eingetragen werden und die alten Geheimtexte zunächst noch bestehen bleiben. Erst wenn die Datenbank komplett neu verschlüsselt ist, werden die jeweiligen Spalten getauscht, was bei relationalen Datenbanken recht flott geht. Dann werden die alten Geheimtexte gelöscht und auch der alte Zeitschlüssel. Das Konzept eines regelmäßig zu wechselnden Schlüssels ist generell nichts Neues. Es mag daher auch noch weitere Verfahren geben die hierfür (Umschlüsselung) geeignet sind.

Dem Zeitschlüssel wird jeweils ein grober Hinweis auf sein Erstellungsdatum hinzugefügt (siehe b). Die festgelegte Schlüsselgröße soll dabei nicht überschritten werden.

Entstehung und Speicherung gilt ansonsten analog zum Masterkey, wobei bei der Entstehung (erstmalige Erzeugung nach Installation bzw. nach einem System-Neustart) einfach eine nicht-deterministische kryptografische Zufallszahl (siehe c) in entsprechender Größe erzeugt wird.

G. BESCHREIBUNG ANSPRUCH 5: (Hashing & Verschlüsselung von Passwörtern)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
zum erweiterten Schutz von Passwörtern und/oder Anmeldenamen, diese bereits bei oder unmittelbar nach der Eingabe, mit einem kryptologischen Einweg-Hashing in Hash-Werte gewandelt werden , welche ganz oder teilweise miteinander kombiniert und zum Teil verkürzt / komprimiert werden sodass eine Rückrechnung unmöglich ist und während der Übermittlung, zusätzlich zu einer etwaigen Transportverschlüsselung wie z.B. bei HTTPS, durch mindestens einen weiteren Geheimschlüssel und mindestens einen vom Empfänger des Passworts übersandten Code verschlüsselt werden, wobei der auf beiden Seiten (Client und Server) bekannte Passwort-Hash und/oder Anmeldename wesentlich zur Generierung der Geheimschlüssel und/oder Codes mit herangezogen wird. (Fig. 4)

Anmeldedaten (Login-Daten) bestehen i.d.R. und so soll dies hier als angenommen gelten, aus einem Anmeldenamen (Login-Namen) und einem Passwort. Diese unterliegen einem besonderen Schutzbedürfnis, insbesondere jedoch die Passwörter. Die meisten Anwender nutzen dieselben Anmeldedaten und Passwörter für alle oder zumindest mehrere Dienste. Ein erbeutetes Passwort kann im Extremfall zum unberechtigten Zugang zu allen Online-Diensten des Anwenders führen, inklusive eigener Cloud, Smart-Home und Online-Banking.

Die Gefahren bestehen im Groben zum einen im sog. Klartext-Lauschen/Mithören der Übertragung (live, d.h. evtl. Verschlüsselung wurde gebrochen) zwischen passworteingebenden Client und dem Server zu dessen Dienst ein Zugang gewährt werden soll und zum anderen im deutlich einfacheren Mitschneiden einer solchen Übertragung, um sie später „in Ruhe“ knacken / auswerten zu können. Dann gibt es noch die Gefahr dass der Client-Rechner gekapert wird (z.B. mit Trojaner, Bot, Viren oder sonstigen Schadprogrammen) um das Passwort oder Schlüssel dort zu erbeuten und den Angriff auf die Server direkt um die dort gespeicherten Daten zu stehlen. Ein Brute-Force-Angriff (Durchprobieren aller Schlüssel) auf den Server ist auszuschließen da mittlerweile auch die letzte Server-Applikation eine zeitliche Begrenzung von Anmeldeversuchen verbaut haben dürfte.
Insgesamt wäre es für den Angreifer traumhaft, das Passwort als Klartext zu ergattern, was immer noch erschreckend oft passiert, doch meist gibt man sich auch mit einem Passwort-Hash zufrieden. Dieser wird dann entweder in Simulation eines Client an den Server übertragen oder es wird versucht damit das Passwort oder ein Pseudo-Passwort (das genauso funktioniert weil es

denselben Hash hervorbringt) zu finden, was teilweise doch überraschend einfach und schnell gelingt.

Die Intensität der jeweiligen Schutzmechanismen muss der potentiellen Gefahr daraus angepasst sein.

1. Passwort-Hashing - Stand der Technik

Um die Erbeutung eines Passwort-Hash aus der Datenbank heraus zu verhindern, gibt es das vorliegende Verfahren mit den bisher bekannten und noch folgenden Verfahrensschritten. Um den Diebstahl des Passwortes in Klartext völlig auszuschließen werden, als Teil des vorliegenden Verfahrens, wie im Anspruch 5 definiert, Klartext-Passwörter nie komplett dauerhaft gespeichert. Auch nicht „nur“ verschlüsselt. Vielmehr werden Passwörter, wie man sie z.B. bei einer Registrierung (erstmalige Anmeldung) an z.B. einer Website bzw. einem Web-Server oder aber auch am Dialogfenster zu einer Anwendung wie z.B. einer Datenbank-Anwendung oder einem eMail-Client) eingibt, möglichst sofort, d.h. vom Browser / Frontend / Dialogfenster direkt, mit oder aber spätestens nach deren Eingabe, allerspätestens jedoch vom Web-Server nach deren sicheren Übertragung dorthin, per kryptologischen Einweg-Hashings (OWHF) und dann ein Teil davon per Prüfsummen-Hashing in (insgesamt) 25% verkürzte HASH-WERTE gewandelt, sodass die Passwörter nicht rekonstruiert werden können. Nur diese Hash-Werte werden (verschlüsselt) gespeichert. Was innerhalb des vorliegenden Verfahrens angewandt natürlich mit nochmaliger (2-facher) Verschlüsselung, wie bereits bis hier insbesondere in Anspruch 4 und nachfolgend dargestellt, geschützt geschieht. Alle Speicher-Rückstände vom Klartext-Passwort und dessen Eingabe werden sicher gelöscht. (OWHF bedeutet One-Way Hash Function und steht für die Erstellung eines Hash-Wertes aus dem selbst bei Bekanntheit des benutzten Verfahrens/Algorithmus/Schlüssels eine Wiederherstellung des Ausgangswertes (Eingabewert, hier also das Passwort) unmöglich ist. Das nennt man eine Einweg-Hashfunktion.) Bei späteren Passworteingaben (nachfolgende Anmeldungen) wird - bis auf das letztendliche Speichern - das selbe Verfahren angewendet und dann nur noch der neu erzeugte Hash mit dem in der Datenbank gespeicherten (ggf. nach dessen Entschlüsselung) verglichen.

Dies ist eigentlich Stand der Technik und sollte von jedem Dienst der Passwörter abfragt auch so genutzt werden. Es wird hier aufgeführt, da es zum Gesamtkonzept des vorliegenden Datenschutz-Verfahrens mit dazu gehört, doch vor allem, weil es die Basis für die nachfolgenden Erweiterungen ist, welche entwickelt wurden um den bisher gegebenen - aber auch schon gebrochenen - Stand-der-Technik-Schutz auszubauen.

Die Verschlüsselung bzw. das Hashen eines Passworts allein reicht eben nicht aus um absolut sicher gehen zu können dass es nicht erbeutet werden kann. Dazu gehört ein Gesamtkonzept, welches direkt bei der Eingabe beginnt, sich bei der Übermittlung fortsetzt und dann erst in der (nochmalig verschlüsselten) Speicherung endet. Und, nicht nur die heute gegebenen Möglichkeiten berücksichtigt.

Aus diesem Grund stellt die vorliegende Erfindung auch Lösungen für

- die richtige Verschlüsselung von Passwörtern,
- die sichere Übermittlung von Passwörtern und
- die abhörsichere Eingabe von Passwörtern

bereit.

2. Verschlüsselung von Passwörtern

Wie erwähnt, sollte das Hashen von Passwörtern Standard sein. Dabei gibt es u.a. das kryptologische Hashen (speziell auch für Passwörter) was zumeist Ergebniswerte >= 256 Bit hervor bringt oder das verkürzende Hashen was i.d.R. zur Verifizierung von Daten-Übertragungen eingesetzt wird.

Diese Verkürzung hat den Vorteil dass Informationen „verloren“ gehen und es daher selbst theoretisch nicht machbar ist, den Ausgangswert wieder zu berechnen. Jedoch gibt es immer die Möglichkeit diesen Ausgangswert zu erraten bzw. alle möglichen Eingaben durch zu probieren bis ein Wert gefunden wurde der denselben Hashwert ergibt. Man nennt das Wörterbuch- bzw. Brute-Force-Angriff. Und damit sind wir auch schon beim großen Nachteil eines verkürzten Hash-Wertes: Es braucht nicht viel Zeit, ein Pseudo-Passwort zu finden, welches mit dem womöglich bekannten Hash-Verfahren denselben Ergebnis-Hash erbringt.

Also doch einen langen Hashwert? Es gibt kryptologische Hash-Funktionen die mindestens genauso wenig berechenbar sind wie gute Verschlüsselungen. Dennoch, - wen man sich mit der technischen Weiterentwicklung (siehe K.4 Technischer Fortschritt) vertraut macht, stellt man fest, dass das was heute noch als sicher gilt schon in 20 Jahren lächerlich sein könnte. Dann könnten erbeutete Passwörter irgendwann doch herausgefunden werden und das ist in jedem Fall zu vermeiden. Aus diesem Grund wählt die vorliegende Erfindung eine Kombination aus Beidem. Es wird zunächst ein expandierendes Hash-Verfahren angewandt und dann ein Teil davon komprimiert/verkürzt. (siehe Pos. 1-3, Fig. 4) Dazu später mehr.

Zwar wird das Passwort (egal wie es gehasht wurde) vom vorliegenden Datenschutzverfahren ausreichend sicher verschlüsselt gespeichert sodass eine Entwendung von da an ausgeschlossen

werden kann, doch kann ein Passwort bzw. der Hash davon auch schon auf dem Weg dahin erbeutet werden.

Dies zu verhindern bzw. zu erschweren ist Thema der nachfolgenden Unterkapitel.

Das Problem bei allen Passwort-Chiffren: Die meisten Anwender geben zu kurze Passwörter ein. Aber selbst bei einem 10-stelligen Passwort ergeben sich (je nachdem welche Sonderzeichen zugelassen werden) „nur“ 80^10 Möglichkeiten. Das entspricht etwa 1e19 oder 2^63. Ein 6 Zeichen Passwort bietet gar nur 2e11 bzw. 2^38. Letzteres ist bei einer recht schwachen Rechenleistung von 1 ns/Durchgang in 4 Minute durch. Aus diesem Grund wurden die Passwort-Hash-Funktionen wie bcrypt oder scrypt entwickelt welche bewusst mit PC-Ressourcen-Zehrenden Funktionen die Berechnung jedes Durchgangs verzögern. Doch angenommen es wird so ausgelegt, dass jede Berechnung bei einem normalen PC etwa 0,5 Sekunden benötigt, dann schafft das in 11 Jahren ein Super-Computer in 1 Stunde alle 2^38 Möglichkeiten. Von speziellen Code-Brech-Maschinen mal ganz abgesehen. Kein wirklich dauerhafter Schutz! Hinzu kommt das Dilemma das die meisten Anwender, normale Wörter als Basis ihrer Passwörter nehmen. Geht man von einem Wortschatz von 50.000 Wörtern aus und nimmt einige beliebte Zahlenkombinationen mit dazu dann liegen wir bei einer Quellmenge von ca. 1 Million Kombinationen. Das hört sich im ersten Moment nach viel an, ist aber für einen Computer nichts. Zudem gibt es besonders beliebte Worte bzw. Kombinationen. In sehr vielen Fällen ist dadurch ein zu dem Hashwert passendes Passwort in unter einer Sekunde zu finden.

Aus diesem Grund ist der Erfinder auch nicht sonderlich überzeugt von den Zeitzehrenden Passwort-Hash-Funktionen wie scrypt oder bcrypt. Legt man diese so aus, dass ein normaler PC 1 Sekunde pro Hashing braucht, dann spielt ein Super-Computer oder ein großes Netzwerk aus normalen PC’s die ganze Liste wahrscheinlicher Passwörter in unter 1 Minute durch. Eine solche Verzögerung bringt erst etwas ab einer Quellmenge (Summe aller durchzuspielenden Möglichkeiten) von größer als 1e13. Wobei dieser Wert alle 10 Jahre um 3 Zehnerpotenzen zunehmen muss. Sollen evtl. erbeutete Passwörter auch in 30 Jahren noch sicher sein dann braucht es schon 1e22 Möglichkeiten. Doch so viele potentielle Passwörter gibt es einfach nicht. Es muss also verhindert werden, dass ein Passwort-Hash überhaupt erst in falsche Hände gerät, denn ist er mal erbeutet kann er „relativ“ leicht geknackt werden. Daher muss er zusätzlich verschlüsselt werden. Dies geschieht bei https-Verbindungen automatisch, doch leider sind diese auch nicht frei von erfolgreichen Angriffen.

Daher verfährt das vorliegende Verfahren nach einer Doppel-Strategie: Zum Einen werden zwei zusätzliche Verschlüsselungen als Schutz gegen das Erbeuten von Passwörtern eingebaut und zum Zweiten wird das Erbeuten der dafür verwendeten Schlüssel besonders erschwert.

3. Sicherheit der Übertragung von (gehashten) Passwörtern

Dummerweise gibt es derzeit keinen wirklich sicheren Übertragungsstandard und es kann heute noch niemand sagen ob das aktuell als weitgehend sicher geltende https-Verfahren (nur inkl. aller sicherheitsrelevanten Neuerungen / Erweiterungen) auch in Zukunft als sicher angesehen werden kann. Hinzu kommt, dass für manche Fälle „weitgehend sicher“ nicht ausreicht. Durch die NSA-Affäre ist bekannt geworden, dass es der NSA gelang sich in https-Übertragungen einzuklinken bzw. solche abzuhören. Die vorliegende Erfindung stellt daher auch Lösungen für die Problematiken der Übertragung bzw. Kommunikation von Daten zur Verfügung.

Wir gehen hier davon aus, ein Anwender am Client-PC hat Anmeldenamen und Passwort eingegeben. Der Anmeldename (AN) wurde bereits mit einer kryptologischen und kollisionsresistenten Hashfunktion in einen Standard-AN-Hash gewandelt und per https-Verbindung an den Server gesendet. Das Passwort noch nicht. (Pos. 1, Fig. 3 und Pos. 0, Fig. 4)
Der Server identifiziert das Konto des Anwenders anhand des Standard-AN-Hash der ebenso in der Datenbank als Anmeldename gespeichert ist. Im vorliegenden Verfahren muss dieser Hash vorher noch mit dem unbearbeiteten Masterkey und Zeitschlüssel verschlüsselt werden, damit die Datenbank-Suche erfolgreich sein kann.
Zusätzlich wird aus dem Anmeldenamen ein 512-Bit-Einweg-Hashwert gebildet. Dieser wird nur bei der Erst-Anmeldung (Registrierung) ebenfalls an den Server übertragen. Bei allen nachfolgenden Anmeldungen darf dies nicht mehr geschehen da dieser Wert als beiderseitiges Geheimnis hilft, das Passwort sicher zu übermitteln. (Pos. 0, Fig. 4)
Das Passwort wird sogleich mit Eigendaten auf 16 Zeichen aufgefüllt und mit einem weiteren (aufwandstechnisch vergleichsweise einfachen) Hashwert aus dem Anmeldenamen kombiniert (Pos. 1, Fig. 4) und per kryptologischen (Passwort-)Hashing (z.B. bcrypt, scrypt, solange DHM unbrechbar ist, würde auch SHA-3, SHA-256, Whirlpool ausreichen, wird aber in Hinblick auf die weitere technische Entwicklung nicht empfohlen) in einen 256-Bit-Hash gewandelt. (Pos. 2, Fig. 4). Sodann wird das Klartext-Passwort überall sicher gelöscht, ebenso wie der Klartext-Anmeldename. Der 256-Bit-Hash wird nun in 2 128-Bit-Hash-Teile gesplittet, wobei die unteren 128-Bit per verkürzenden Hashing (z.B. CRC-64) auf 64 Bit komprimiert werden. (Pos. 3, Fig. 4) Diese 64 Bit werden nun als höherwertige 64 Bit des zweiten (niederwertigen) 128-Bit-Hash-Teils genommen und als niederwertige 64-Bit wird eine 32-Bit-IP (high) und ein 32-Bit-Zeitstempel (low) dazu genommen. Sodass wieder ein vollständiger 256-Bit-Wert entsteht, den wir Multicode nennen.

Nun wird innerhalb einer sichern (https-)Verbindung mit Hilfe des Diffie-Hellman-Merkle-Schlüsselaustausch (DHM) ein gemeinsamer geheimer Schlüssel geschaffen, welcher nun dem Server und dem Client zur Verfügung steht ohne dass er übermittelt werden musste.
DHM gilt jedoch definitiv nicht als unbrechbar. Vor allem dann, wenn die Grundparameter (Primzahl, Generator) bekannt sind (was i.d.R. der Fall ist da diese gegenseitig ausgetauscht oder für den jeweiligen Server weitgehend geläufig sind) und sich diese in Größenordnungen von 300-stelligen Zahlen oder weniger bewegen, scheint es für Hochleistungsrechner ein Leichtes zu sein, den DHM-Schlüssel zu finden. Aufgrund des hohen Aufwandes so große (Prim-)Zahlen zu erschaffen werden oft immer die gleichen Werte benutzt oder ein Wertepaar aus einer vorgegebenen relativ kleinen Menge. Aber auch das Neu-Kreieren einer z.B. 500-stelligen Primzahl bringt keinen großen Sicherheitsgewinn, wenn diese anschließend über eine abgehörte Leitung geschickt wird. Natürlich ist das DHM genau dafür geschaffen worden, dass ein geheimer Schlüssel entsteht obwohl die Parameter und auch der öffentliche Schlüssel bekannt sein dürfen, aber die Sicherheit ist leider nun mal dann nicht sonderlich hoch. Das ist der Schwachpunkt dieses Verfahrens. Das BSI empfiehlt daher Zahlen mit mindestens 600 Dezimalstellen zu verwenden. Doch auch dies ist nur eine Frage der Zeit.

a) Der Trick mit der geheimen Primzahl

Aus diesem Grund - und auch um die Gefahr eines sog. Man-In-The-Middle-Angriffs abzuwehren -wenden wir hier zu Erzeugung der Primzahl ein Geheimnis an, das nur Empfänger und Sender kennen ohne dass es kürzlich übertragen wurde (und somit mitgehörte werden konnte). Das Passwort, bzw. dessen Hash und der sog. Generator-Hash aus dem Anmeldenamen. (Außer natürlich bei der Erstregistrierung. Doch wie groß ist schon die Wahrscheinlichkeit dass ein Angriff erfolgt auf ein Konto das noch gar nicht existiert bzw. gerade dabei ist, angelegt zu werden, und dann zwar die Primzahl vom Server generiert und zum Client übertragen wird, aber somit immer noch die Sicherheit eines hochwertigen DHM besteht, zumal man für die Registrierung einmal die Größenordnung der Grundparameter mindestens verdoppeln kann oder per 2-Stufen-Autentifizierung eine zusätzliche Sicherheit einbringen kann.) Zur Erzeugung der Primzahl wird der obere (linke) 128-Bit-Teil des Passwort-Hash heran gezogen. (Pos. 4, Fig. 4) Dabei wird folgende Rechnung angewandt:

MP = 400 / Stellen_PWH + Stelle2_PWH * 1,5

Stellen_PWH = Anzahl 10er Potenzen des Passwort-Hash. (Dürfte ca. 38 sein), Stelle2_PWH = 2te Dezimalstelle des Passwort-Hash (PWH). Danach wird ausgeführt: PZB = PWH ^ MP

Das Ergebnis (PZB) ist eine Zahl mit 404 bis 924 Stellen, aus welcher die nächstgrößere oder nächstkleinere Primzahl zu finden ist.

Natürlich gibt es für diese große Zahl nicht so viel Möglichkeiten wie bei einer frei erfundenen, denn letztlich kann man beim Brute-Force-Durchprobieren, zu dem resultierenden Passwort-Hash auch gleich die jeweilige Primzahl berechnen, doch das bedeutet auch, um den DHM-Schlüssel zu knacken müssen nicht etwa nur 10 Primzahlen oder gar nur eine (wenn der https-Schlüssel gebrochen wurde, konnte man die DHM-Grundwerte ja mitlauschen), sondern eben doch beispielsweise 1,6e15 also etwa 100 Billionen mal so viel getestet werden. Und selbst wenn man die richtige Primzahl gefunden hätte, ist der DHM-Schlüssel noch nicht gebrochen. Dafür braucht dann auch die NSA mal so 3 Millionen Jahre. Na gut, - man wird sicher einen Erleichterungs-Algorithmus finden so dass es nur ein paar 1000 Jahre dauert, aber bis dahin kann die Größe der Primzahl sich mit der Entwicklung der Rechenleistung Stück für Stück anpassen. Die Primzahl ist nun also auf beiden Seiten bekannt ohne dass sie übertragen oder festgelegt werden musste. Anschließend überträgt der Server einen Teil des Generators. Ein weiterer Teil könnte aus den Informationen des Server-Zertifikates entnommen werden und ein weiterer Teil entsteht aus dem oben bereits erwähnten Generator-Hash. Dieser Hash, welcher aus dem Anmeldenamen jedes Mal auf Client-Seite einfach zu berechnen und auf Server-Seite aus der Registrierung bekannt (gespeichert) ist, dient als weiteres Geheimnis welches ausschließlich die Kommunikationspartner wissen. Dieser Generator-Hash wird nun mit dem in Pos. 3 Fig. 4 gebildeten 64-Bit-Hash (CRC-64-Hash aus unteren 128 Bit des 256-Bit -Hash aus Passwort, Erweiterung auf 16 Byte und einfach gehashten Anmeldenamen) verschlüsselt. Das Ergebnis dient zusammen mit den Teilen aus Zertifikat-Information und vom Server übertragenen Teil (Welcher so angepasst ist dass der Generator die mathematischen Voraussetzungen für das DHM-Verfahren erfüllt.) als Generator für das DHM-Verfahren, welches nun ausgeführt werden kann. Es entsteht ein auf beiden Seiten bekannter aber geheimer Schlüssel den selbst ein mitlauschender nicht kennen und brechen kann. (Pos. 4, Fig. 4)

Sollte sich die Berechnung der Primzahl auf den meisten Systemen als zu zeitaufwendig heraus stellen, so ist das o.g. Verfahren zu erleichtern, was für solche Systeme und deren Angriffs-Ressourcen bei weitem noch ausreichen dürfte. Es gäbe auch die Möglichkeit die Berechnungen zu tauschen, sodass der Client anstatt der Primzahl den Generator aus dem 128-Bit-Teil des Passwort-Hash berechnet und die Primzahl aus den 3 Teilen zusammen gefügt wird. Die o.g. Berechnungsformel ist entsprechend anzupassen. Zusätzlich kann die Umstellung von DHM auf die Basis mit Elliptischen Kurven eine deutliche Rechenerleichterung bringen da sich die Anzahl der Parameter-Längen bei gleicher Sicherheit um den Faktor 10 reduziert.

Alternativ kann der Server mithelfen. Er muss die Primzahl ja selbst auch berechnen und könnte dem Client dann Hilfestellungen geben, die richtige Zahl deutlich schneller zu finden. Beispielsweise eine Zahl um die der berechnete Exponent vom 128-Bit-Hash (MP) erhöht werden muss um gleich viel näher am nächsten Primwert zu liegen. Ist dies ein Bruchwert kann man schon recht nah ran kommen. Dieser Wert könnte noch weiter verschleiert werden indem er z.B. auch noch zu den Ziffern 7 - 12 des 128-Bit-Hash addiert werden muss. Zusätzlich könnten weitere Relativ-Werte übermittelt werden um dann ganz genau den Primwert zu treffen ohne ihn selbst testen zu müssen. Oder eine Prüfsumme die es dem Client erlaubt über einige wenige Versuche zu dem tatsächlichen Primwert zu gelangen. Da die Hinweise stets relativ sind und/oder bezogen auf den ansonsten unbekannten Ausgangswert nützen sie einem Angreifer recht wenig. Dennoch sollten solche Hilfestellungen mit einer erneuten asymmetrischen Verschlüsselung übertragen werden, wenn der Zeitrahmen es zulässt.
Dies gilt ebenso für die nachfolgende alternative Hilfestellung.
Diese könnte so ablaufen dass der Server etwa z.B. 10.000 Primzahlen in der Länge von 400 - 900 Dezimalstellen übermittelt. Das hört sich aufwendig an, sind aber nur 3 MB und geht heutzutage innerhalb von 1 Sekunde. Dann braucht der Client nur die raus suchen die am nahesten an der Zahl ist die mit o.g. Rechnung berechnet wurde. Diese Primzahlen müssen natürlich variieren. Sie jedes Mal frisch zu berechnen wäre jedoch zu aufwendig. Die praxisnähere Alternative wäre sie aus einem vorberechneten Pool von mindestens 100 Mio. heraus zu fischen. Dieser Pool könnte zur Sicherheit immer wieder neue Zahlen selbständig im Hintergrund generieren und andere damit ersetzen so dass auch hier ein gewisser Wechsel stattfindet. Selbstverständlich müsste dieser Pool entsprechend geschützt und verschlüsselt sein.

b) Multicode - doppelt verschlüsselt hält besser

Nun sendet der Server; an dem ein Passwort übermittelt werden soll, eine, mit dem eben geschaffenen DHM-Geheimschlüssel verschlüsselte, 128-Bit-Zufallszahl an den Client. Dies macht der Server pro Anmeldename und IP nur z.B. dreimal innerhalb von 5 Min. und maximal 10-mal in einer Stunde, womit Wörterbuch- und Brute-Force-Angriffe auf den Server direkt schon ausgeschlossen sind (Begrenzung der Anmeldeversuche).
Der Client entschlüsselt mit seinem DHM-Geheimschlüssel den 128-Bit-Code (Pos. 5, Fig. 4) und verschlüsselt sogleich die so gewonnene Zufallszahl mit den oberen 128-Bit des Passwort-Hash bzw. des Multicodes und erhält einen 128-Bit-Schlüssel (Pos. 6, Fig. 4) und löscht sicher die Zufallszahl sowie den empfangenen 128-Bit-Code. Zur Verschlüsselung kann gerne AES oder eine andere hochwertige / sichere Methode zum Einsatz kommen.

Mit diesem 128-Bit-Schlüssel wird nun der oben generierte Multicode verschlüsselt. (Pos. 7, Fig. 4) Dies ist notwendig um einen Angriff auf den DHM-Schlüssel massiv zu erschweren. Zuguterletzt wird nun mit dem geheimen DHM-Schlüssel (per AES oder vergleichbarem) der verschlüsselte Multicode noch ein weiteres mal verschlüsselt und dann an den Server gesendet. (Pos. 8, Fig. 4) Da der geheime DHM-Schlüssel nur für diese eine Passwort-Übermittlung zu verwenden ist, wird dieser sogleich sicher gelöscht, genauso wie der 128-Bit-Schlüssel.
(Die jeweils genannten Schlüssel- und Hash-Größen sind nur beispielhaft. Sie sollten jedoch nicht unterschritten, können aber gerne vergrößert werden, insbesondere bei besonders sensiblen Daten)
Wichtig ist, dass für die Verschlüsselung des Multicode mit dem 128-Bit-Schlüssel ein anderes Verschlüsselungsverfahren eingesetzt wird als es für die Verschlüsselung mit dem DHM-Schlüssel verwendet wird.

Der Server wiederum entschlüsselt zuerst mit dem geheimen DHM-Schlüssel und dann mit dem selbst generierten 128-Bit-Schlüssel (aus dem ihm vorliegenden Passwort-Hash und dem gesendeten Zufallswert) den Multicode. Dann prüft er ob IP stimmen und der Zeitstempel in einem akzeptablen Rahmen liegt (in Bezug auf die Jetzt-Zeit und in Bezug auf die Sendezeit der Zufallszahl und des öffentlichen DHM-Schlüssels). Falls ja werden die oberen 192 Bit des Multicode als Passwort-Hash (mit Masterkey und Zeitschlüssel) verschlüsselt und mit dem vorhandenen verschlüsselten Passwort-Hash verglichen (nachfolgende Anmeldung).

Standardmäßig löscht der Server alle Sendeaufträge nach 5 Minuten inklusive des geheimen DHM-Schlüssels, so dass verspätete oder unaufgeforderte Multicodes ins Leere laufen.

Bei der Registrierung (Erstanmeldung) ist der Passwort-Hash auf Server-Seite noch nicht bekannt. Daher muss - wie bereits erwähnt - der Primwert an den Client übermittelt werden oder es wird ein bekannter oder einer aus dem Zertifikat des Servers benutzt. Zur Übermittlung sollte eine starke asymmetrische Verschlüsselung (zusätzlich zur https-Verschlüsselung) genutzt werden. Die erste Verschlüsselungsstufe mit dem Multicode (Pos. 7, Fig. 4) fällt dann entweder weg oder wird mit dem eben erwähnten asymmetrischen Schlüssel vollzogen.

Selbst wenn ein Angreifer per Brute-Force/Wörterbuch die oberen 192-Bit des Multicode finden würde und eine IP simulieren sowie einen Zeitstempel generieren würde, fehlt ihm dann noch die Zufallszahl um den simulierten Multicode zu verschlüsseln. Ein einfacher Brute-Force darauf schlägt fehl da er das Ergebnis nicht auf Korrektheit testen kann. Ferner müsste für eine Übertragung an den Server noch der passende DHM-Schlüssel vorliegen, doch auch hier kann das Ergebnis nur sehr aufwendig (mit jedes Mal 2^50 Möglichkeit) auf Korrektheit getestet

werden.

Selbst wenn ein Angreifer die Kommunikation abfangen könnte und die https-Verschlüsselung geknackt hat, müsste er nun noch den geheimen DHM-Schlüssel finden um an den zuvor gesendeten Zufallswert heran zu kommen. Jetzt könnte erst der Brute-Force-Angriff beginnen um ein Pseudo-Passwort zu finden, welches den 128-Bit-Hash ergibt mit dem der Zufallswert verschlüsselt den kompletten Hash errechnet. Berechnen kann man es jedenfalls nicht. Und - was einer der wesentlichen Aspekte ist, es ist nicht möglich sich als Client auszugeben und einfach einen abgefangenen 256-Bit-Hash an den Server zu senden so als wäre gerade das korrekte Passwort eingegeben worden, denn dieser ist jedes Mal anders und erfordert zunächst eine Anforderung vom Server. Eine Simulation ist jedoch nur möglich wenn tatsächlich zuvor beide (https und DHM) Verschlüsselungen gebrochen wurden, und - was heute Standard ist, die IP zu den bisherigen Anmeldungen stimmt oder wenn nicht, die Rechner-/Geräte-ID stimmt. Andernfalls wird eine 2-Faktor-Authentifizierung gefordert.

Der entscheidende Unterschied und Vorteil in diesem Verfahren liegt

- in der enormen Verbesserung der Sicherheit des DHM durch zwei geheime Grund-Parameter - Primzahl und Generator, und
- in der doppelten Verschlüsselung die einen Angriff auf den DHM unmöglich macht da das Ergebnis nicht einfach aus einer relativ kleinen Menge an Passwort-Hash's getestet werden kann. Es müssen zu jedem potentiellen DHM-Schlüssel alle möglichen Passwort-Hash's (im Schnitt 80^8) und Anmeldenamen (im Schnitt 40^15) durchprobiert werden.

Eine Brechung des DHM-Schlüssels kann somit ausgeschlossen werden.

Der Angreifer müsste also nach geknackter https-Verschlüsselung für alle möglichen Passwörter die Primzahl berechnen und für alle möglichen Anmeldenamen den Generator und damit dann versuchen den DHM-Schlüssel zu knacken, was bei der empfohlenen Schlüsselgröße noch nicht gelungen ist. Doch selbst bei nur 512 Bit Schlüsselgröße wäre es eine Rechenarbeit von > 50 Millionen Jahre vorausgesetzt man ist 7-mal so schnell wie die bekannten Erfolge der NSA und der Anmeldename ist bekannt. Ansonsten dauert es noch 1e24 mal so lange.

So geschützte Passwörter sollten somit die nächsten 100 Jahre sicher sein.

Ggf. könnte zur nochmaligen verschlüsselten Übertragung des Generator-Teils und der öffentl. Schlüssel des DHM-Verfahrens ein asymmetrischer Schlüssel vom Client an den Server geliefert werden. Aber so lange das vorliegende Verfahren - wie aktuell - nicht gebrochen werden kann, sollte es so ausreichen.

c) Hashing des Anmeldenamens

Es wird hier davon ausgegangen, dass der Anmeldename in der jeweiligen (Web-)Applikation / Anwendung nicht in Klartext benötigt wird und so die Sicherheit der Passwort-Übermittlung durch die in Pos. 0 Fig. 4 dargestellten Schritte - wie oben ausgeführt - erheblich gesteigert werden kann. Sollte ein Klartext-Name erforderlich sein, wäre es sinnvoll einen solchen (Alias, Nicknamen oder Rufnamen) zusätzlich abzufragen.

In vorliegender Ausgestaltung der Erfindung wird der Anmeldename ähnlich wie das Passwort kryptologisch gehasht. Im Unterschied zu dem Verfahren für das Passwort-Hashing muss jedoch dieses kollisionsresistent sein. D.h. es darf keine zwei Eingangswerte geben die zu dem selben Ausgangswert führen. Da nun der Anmeldename in Klartext nicht übertragen und gespeichert wird, kann dieser in mindestens 2 Bereichen die Sicherheit steigern:

- Ein zusätzlicher Hashwert daraus, generiert entweder mit anderem Schlüssel oder anderem Verfahren als das für den Hashwert der an den Server übermittelt wird, kann benutzt werden den Generator für das DHM zu generieren oder kann als Schlüssel benutzt werden um dessen Übermittlung oder die Übermittlung des öffentlichen DHM-Schlüssels oder der DHM-Primzahl oder Informationen dazu zusätzlich zu verschlüsseln. Der DHM-Generator wird dabei aus 3 oder 4 Teilen zusammen gefügt: Einen festen Teil, einen Teil der vom Server an den Client übertragen wurde, einen Teil den der Client aus dem Zertifikat des Servers entnommen hat und dem o.g. zusätzlichen Hashwert.
- Der Anmeldename in Klartext oder ein zusätzlicher Hashwert daraus, generiert entweder mit anderem Schlüssel oder anderem Verfahren als das für den Hashwert der an den Server übermittelt wird, kann zu dem Klartext-Passwort (und ggf. dessen Erweiterung auf 16 Zeichen) hinzugefügt oder damit multipliziert werden, bevor dieses wie oben dargestellt kryptologisch gehasht wird. Somit ist ein Wörter-Buch-Angriff nahezu ausgeschlossen oder zumindest massiv erschwert, selbst wenn der Multicode-Hash entschlüsselt erbeutet werden könnte.

Weitere Sicherheitsmaßnahmen wie die 2-Stufen-Authentifizierung vor allem bei Anmeldeversuchen über ein unbekanntes Gerät sind Stand der Technik und sollten entsprechend berücksichtigt werden.

Natürlich kann dieses erweiterte Schutzverfahren nicht nur auf Passwörter angewandt werden. Da der Rechenaufwand jedoch sehr hoch ist, wird es wohl noch eine Zeit lang wirklich sensiblen Daten vorbehalten sein. Mit steigender Rechenleistung muss dann allerdings auch das Verfahren

angepasst werden. Dafür genügt dann aber eine Erhöhung der Schlüsselgrößen oder ggf. eine zusätzliche Verschachtelung, so dass z.B. mit dem ersten DHM-Schlüssel nur erst mal Grundwerte für ein zweites DHM-Verfahren übertragen werden, oder ein weiterer asymmetrischer Schlüssel. Man muss dann jeweils sehen was am effizientesten schützt und mit den gegebenen Rechenleistungen anwendbar ist.

Da der so aufwendig generierte DHM-Schlüssel nach der Übermittlung des Passwortes ja nun da ist, könnte die Verführung groß sein, auch weitere Daten damit zu verschlüsseln. Doch dabei sollte bedacht werden, dass die womöglich eine Krypto-Analyse vereinfacht. Besser wäre dann mit dem DHM-Schlüssel einen weiteren symmetrischen Schlüssel auszutauschen und den DHM-Schlüssel zu löschen. Somit könnte eine Krypto-Analyse nur diesen neuen Schlüssel brechen, nicht aber den DHM-Schlüssel der das Passwort schützt.

4. Eingabe von Passwörtern

a) Spezielle Eingabefelder

Das jeweils angewandte Hashing (Pos. 0-3, Fig. 4) sollte so früh und so nativ wie möglich angewandt werden.

Idealerweise wird die Hashfunktion bereits im Eingabefeld integriert. Dazu muss beispielsweise ein eigenes Eingabefeld-Objekt programmiert werden. Zwar gibt es bei den meisten Systemen bereits Eingabefelder welche die Eingabe mit Sternchen verschleiern, doch hier geht es auch darum dass die Eingabe nicht zwischengespeichert wird. Dies ist normalerweise der Fall und der eingegebene Wert wird mit „Enter“ an das Programm übergeben das den Dialog aufgerufen hat. Dies könnte zwar jetzt den übergebenen Wert sofort hashen doch bis da könnte es bereits schon wieder abgefangen worden sein. Insbesondere das Botschaften-System wie z.B. bei Windows macht es einem Angreifer relativ leicht, mitzuhören was bei / in anderen Programmen so passiert. Da - wie wir weiter oben feststellen mussten, ein Hashwert keine besondere Sicherheit darstellt, sollte dieser auch für die Systeminterne Übermittlung verschlüsselt werden, bzw. das o.g. Verfahren (G. 3) im Eingabefeld-Objekt weitgehend integriert sein.

b) Spezielle Bildschirmtastatur

Natürlich können auch die Tastendrücke abgefangen werden und es gab bereits reichlich derartige Angriffe, doch dies kann mit einer Bildschirmtastatur verhindert werden. Dann bliebe für einen Angreifer nur noch die Möglichkeit den Inhalt der Grafikkarte zu erbeuten und per OCR / Texterkennung heraus zu finden welche Buchstaben / Tasten gedrückt wurden. Der

Aufwand dafür ist aber schon ordentlich, zumal Grafikkarten-Treiber solche Möglichkeiten erst mal nicht vorsehen. Erschwert kann dieser Angriff mit Tastatur-Zeichen die für OCR kaum zu erkennen sind. Hier helfen schwache Kontraste, Größen- und Farbewechsel aber vor allem auch Verformungen. Zusätzlich kann aufgrund der Mausklicks(-Koordinaten) auf der Bildschirmtastatur Rückschlüsse gezogen werden weshalb die Tasten nicht im üblichen Tatstatur-Schema angeordnet sein dürften sondern ihre Positionen wechseln müssen .
Eine recht sichere Alternative ist die „Zaubertastatur“: Zunächst sieht man eine Bildschirmtastatur im üblichen Layout jedoch ohne Angabe von Buchstaben auf den Tasten. Durch ständige Änderung der Größe der Tastatur kann kein einfacher Rückschluss aus den Mausklick-Daten entnommen werden. Zusätzlich wäre es denkbar dass sich Tasten „magisch“ an eine neue Position bewegen wenn man mit der Maus nur darüber fährt und es wäre dann die entsprechende Taste auf der normalen Tastatur zu drücken. Ein Wechsel der verschiedenen Darstellungen würde es einem Angreifer besonders schwer machen, die Eingaben zu hacken.

Eine Art Kombination aus Beiden ist das als Teil dieses Verfahrens entwickelte Eingabe-Verschleierungs-Verfahren...

H. BESCHREIBUNG ANSPRUCH 6: (Eingabemaske mit Geheim-Botschaften)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
zur Verschleierung von Eingaben der Anwender vom Computer Anweisungen erhält wie er die bevorstehende Eingabe abzuwandeln hat.

Es werden dem Anwender z.B. auf dem Bildschirm verschiedenste Arten von Hinweisen gegeben, wie er sein eigentliches Passwort zu verschleiern hat. Diese werden in Typ, Art, Gestaltung und Parametern innerhalb bestimmter Grenzen zufällig ausgewählt. So baut das Programm eigenständig Sätze zusammen.
Diese könnten beispielsweise lauten: „Nächste Taste um 3 nach rechts (oder links falls das nicht geht)“ oder „im Alphabet 5 weiter (nach z wieder bei a weiter zählen)“. So entsteht aus einem zu drückenden „s“ ein „g“ und aus einem einzugebenden „p“ ein „u“. Auch könnten Pfeile, Helligkeitsunterschiede, Farben oder sonstige Hinweise auf einer Bildschirmtastatur zeigen, auf

welche Art der Anwender seine nächste Eingabe verschleiern soll. Es könnten Rätsel und Phrasen eingebaut werden, oder Hinweise die sich auf Daten des Anwenders, das Datum, das Wetter oder kürzliche Ereignisse beziehen. Beispiele:

- „Verschieben Sie den nächsten Buchstaben um die zweite Zahl Ihres Geburtstages“
- „Rechnen Sie um die erste Ziffer der heutigen Tagestemperatur im Alphabet zurück“
- „Geben Sie als nächstes einfach so viele sinnlose Buchstaben ein wie Ihr Vorname Buchstaben hat und danach den letzten Buchstaben Ihres Passwortes gefolgt von 3 wahllosen Ziffern und dem vorletzten Buchstaben Ihres Passwortes“

Um zu erschweren dass die Texte irgendwo abgefangen werden, könnten diese (teilweise) als Bild übertragen werden. In dieses können bestimmte Zahlen ggf. ebenfalls als Bild oder als Symbol (statt einer „3“ sieht man eine Hand die 3 Finger zeigt) eingefügt werden. Noch effektiver wäre eine teilweise Sprachausgabe und auch hier wieder eine Kombination wie z.B. Audio: „Verschieben Sie den nächsten Buchstaben im Alphabet um 2 mal die Anzahl von Bildteilen auf denen ein Schild zu sehen ist“.
Die Texte, Bilder und Hinweise können, je nach Vorlieben des Anwenders, nacheinander oder auf einmal ausgegeben werden.

Eingaben nach diesem Verfahren sind nicht viel aufwendiger als eine Bildschirmtastatur mit der Maus zu bedienen, aber sicherer. Eine Tastenanschlag-Aufzeichnung geht somit völlig ins Leere. Selbst wenn das Eingabefeld bzw. die Eingabe und deren Übermittlung an die Anwendung abgefangen wird, droht keine Gefahr, da nur das Programm welches das Passwort abfragt und die Hinweise gegeben hat, weiß wie es aus dem verschleierten Passwort wieder das Richtige macht.
Es gibt Software, die Tastenanschläge verschlüsselt, jedoch ist dies mehr ein Gag für Leichtgläubige denn die Verschlüsselungsstärke bringt echte Hacker nur zum Grinsen.
Zusätzlich besteht immer die Gefahr die Tastenanschläge noch vor der Verschlüsselungssoftware abzufangen, oder die wieder entschlüsselten Eingaben zu ergattern.
Es ist daher essentiell, dass alles ein geschlossenes System ist. D.h. das Programm welches das Passwort an den Server senden soll, gibt die Verschleierungshinweise und ruft das Eingabefeld auf, welches wiederum die Tastatur-Eingaben entgegen nimmt. So werden die verschleierten Eingaben innerhalb desselben Programms quasi in einer Programmzeile entschleiert und in der nächsten sogleich gehasht. Die darauf folgenden Befehle überschreiben nun sofort alle Speicherrückstände zum Klartext-Passwort (sicheres löschen) und fahren mit dem o.g. Verfahren zur Verschlüsselung des Passwort-Hash fort, bis dieses an den Server gesandt ist. In dieser Phase

muss darauf geachtet werden, dass durch Multithreading kein Risiko besteht, denn wie oben bereits ausführlich dargestellt, kann ein Einweg-Hash zwar nicht zurück gerechnet werden aber dennoch durch einen z.B. Wörterbuch-Angriff relativ schnell gefunden werden, weshalb der Hash allein keinesfalls als sicher angesehen werden kann, es sei denn der Anwender gibt ein 20-stelliges Passwort ein, das mit normalen Wörtern wenig zu tun hat.

I. Ablauf des kompletten Verfahrens

Überblick:

1. Passwörter hashen bzw. gehasht empfangen
2. Mit Masterkey verschlüsseln
3. Masterkey arithmetisch bearbeiten
4. Mit Zeitschlüssel verschlüsseln
5. Masterkey und Zeitschlüssel zu Futurekeys verschlüsseln

1. Zunächst werden die Anmeldedaten eingegeben. Diese werden gemäß Anspruch 5 gehasht und Passwörter in Kombination mit dem Hash des Anmeldenamen gehasht, verkürzt und mit geheimen Schlüsseln mehrfach verschlüsselt. (Pos. 1-2, Fig. 3)

   Somit wird sichergestellt, dass Passwörter nicht erbeutet werden könne und somit auch nicht in anderen Anwendungen/Websites verwendet werden können. Auch kann somit kein Klartext-Passwort rekonstruiert werden, das es dem Angreifer erlauben könnte, das Konzept, welches ein Anwender gebraucht um Passwörter zu generieren, zu erkennen. Und zwar selbst dann nicht, falls es dem Angreifer gelingen könnte, die (HTTPS-) Transport-Verschlüsselung zu brechen.

2. Da Hash-Verschlüsselung aber durchaus zu knacken* ist und es auch weitere Daten gibt die schützenswert sind, muss ein (z.B. 192-Bit-) Schlüssel (Masterkey) die Hash-Werte oder auch weitere sensible Daten (die jedoch im Gegensatz zum Passwort komplett wieder herstellbar sein müssen und daher nicht derart gehasht werden können) verschlüsseln. (Pos. 6, Fig. 3)

*) „knacken“ bedeute hier – da wir ein verkürztes Einweg-Hashing anwenden - nicht die Wiederherstellung des Quellwertes sondern die Erzeugung eines Pseudo-Quellwertes der unter dem selben Hashing (selbes Verfahren, selber Algorithmus, selber Schlüssel) zu dem selben Ergebnis (Hash-Wert) führt und somit eine Prüfung positiv durchläuft.

Aber auch eine einzelne zusätzliche Verschlüsselung bietet vor allem auf Dauer keine 100%-ige Sicherheit. Wie sich in der Vergangenheit immer wieder gezeigt hat, werden als sicher geltende Systeme nach oft nur kurzer Zeit doch geknackt. Und dies liegt nicht unbedingt nur an einer zu gering gewählten Schlüssellänge.
Um dieses Rest-Risiko weiter zu minimieren, folgen die nächsten Schritte.

3. Der unter 2. verwendete Masterkey wird, bevor er zur Verschlüsselung von Passwörtern (Hash-Wert aus 1.) eingesetzt wird, arithmetisch bearbeitet. (Pos. 5, Fig. 3)
Als arithmetische Berechnung wäre beispielsweise bei einem 128-Bit-Schlüssel eine einfache Multiplikation des Unique-Key’s mit dem zweiten Byte des Masterkeys multipliziert mit 12, eine Multiplikation der Datenfeldnummer mit dem siebten Byte, eine Addition des sechsten Byte mit der Spaltennummer und eine Addition des fünften Wortes mit einem 16-Bit-Prüfziffer-Hashwert des Klartext-Anmeldenamens, sowie eine Bit-Verschiebung des vierten Wortes um die einstellige Quersumme des dritten und vierten Zeichens des vorangehenden Datenfeldes möglich und selbiges mit dem 2-ten und Spaltennummer-ten Zeichen des übernächsten Datenfeldes. Aber hier können auch viele weitere Kombinationen und Methoden Anwendung finden, und u.U. auch eine weitere Verschlüsselung. Die dafür verwendete Funktion / Formel sollte möglichst anwendungsindividuell sein. D.h. sie wird entweder bei der Installation (zufällig) vom Installationsprogramm erzeugt oder vom Hersteller oder Anwender vorgegeben. Ein gelegentlicher (ebenfalls zufällig auszuwählender) Wechsel der Formel-Parameter wäre förderlich, ist aber aus heutiger Sicht nicht notwendig und wäre auch ein Sicherheitsrisiko wenn die Parameter nicht in der Timer-Hardwareerweiterung manipulationssicher gespeichert würden.
Somit ist gewährleistet dass jeder Datensatz mit einem etwas anderen Schlüssel (Masterkey) verschlüsselt wird, was eine Muster-Erkennung (Mustererkennung-Kryptoanalyse), wie sie zum Errechnen bzw. Hacken des Schlüssels erbeuteter Daten eingesetzt werden kann, nahezu ausschließt und einen möglichen Angriff auf das extrem zeitintensive Brute-Force (das systematische durchprobieren aller möglichen Schlüssel) beschränkt.

Wichtig:
Eine solche Mustererkennung-Kryptoanalyse wäre jedoch nach wie vor möglich bei den

Datenfeldern die mit dem Datensatzweise unveränderten Masterkey verschlüsselt werden (sofern die nachfolgende Stufe gebrochen werden kann). Aus diesem Grund sollte für diese Datenfelder ein anderer Masterkey verwendet werden, damit nicht ein für diese Datenfelder erfolgreich durchgeführter Muster-Erkennungs-Angriff auf die Verschlüsselung der anderen Datenfelder durchgreift.

4. Um jegliche Angriffe nach heutigen Stand unmöglich zu machen wird der zuvor generierte Geheimtext (Masterkey-verschlüsselte (Hash-) Werte) zusätzlich nochmals mit einem Zeitschlüssel verschlüsselt, welcher nur für 1 Stunde bis 1 Tag gültig ist, wobei hierfür ein anderes Verschlüsselungsverfahren verwendet wird als zuvor mit dem Masterkey. (Pos. 6, Fig. 3)
   Damit wird ein evtl. erbeuteter Zeitschlüssel in Kürze wertlos und jeglicher Angriff direkt auf dem Datenbankserver erhält eine zeitliche Brisanz die es einmal mehr unmöglich machen dürfte zum Erfolg zu kommen. Außerdem entsteht ein zeitlicher Bezug zwischen Schlüssel und Daten, welcher u.a. im Kapitel L (Anspruch 7) sehr deutlich zum Tragen kommt.

   So werden alle Passwörter und weitere Daten regelmäßig per gültigen Zeitschlüssel dechiffriert und mit dem neuen sofort wieder chiffriert.

Ein Online-Angriff ist somit aufgrund der jeweils kurzen Zeitfenster unmöglich und wenn die komplette (verschlüsselte) Datenbank erbeutet wird, dürfte deren Dechiffrierung aufgrund der drei- bis vierfachen Verschlüsselungstiefe als (zeitlich) unrealistisch angesehen werden, denn das Ergebnis aus dem ersten Hackversuch (Zeitschlüssel) bringt keinen Klartext der dem Angreifer zeigen könnte, dass er sein Teilziel erreicht hat und es gibt – auch aufgrund der arithmetischen Bearbeitung und der unterschiedlichen Verschlüsselungsverfahren – keinen einzelnen Schlüssel mit dem die gesamte Datenbank dechiffriert werden kann. Die Möglichkeiten steigen von 2^128 (eine 39-stellige Dezimalzahl) auf 2^256, einer Zahl mit 78 Stellen. Würde jede Berechnung nur 1 PikoSekunde (1 millionstel Mikrosekunde) dauern, wäre ein Computer damit 3e57 Jahre beschäftigt wobei das Weltall erst seit 1,4e10 Jahren besteht.
Nun bestünde theoretisch nur noch die Gefahr dass die Schlüssel (Masterkey und Zeitschlüssel) trotz aller üblichen Vorsichtsmaßnahmen (womöglich zusammen mit den zugehörigen (mehrfach verschlüsselten) Daten) erbeutet werden. Aus diesem Grund folgt die nächste Sicherheitsstufe.

Es sei darauf hingewiesen dass eine 256-Bit Verschlüsselung (u.U. irgendwann) deutlich

einfacher zu brechen ist, als eine doppelte Verschlüsselung á 128 Bit mit unterschiedlichen Verfahren und wechselnden Schlüsseln.

5. Es gibt bislang 2 Schlüssel (Ausgangsschlüssel) im System. Den Masterkey und den Zeitschlüssel. Um sicherzustellen, dass diese Schlüssel nicht erbeutet werden können, werden sie generell nicht dauerhaft gespeichert, - weder auf Festplatte noch im Arbeitsspeicher. Stattdessen werden sie mit einem Wert, dem Zeitcode, verschlüsselt der sich aus einem zukünftigen Zeitwert (Zeit X) des Systems (oder eines gesetzten Timers) errechnet. Nur die sich jeweils daraus ergebenden Futurekeys werden - versteckt - gespeichert, die Originale (Masterkey und Zeitschlüssel) werden sofort sicher gelöscht. (Pos. 7, Fig. 3 und Pos. 1-3, Fig. 1) Doch diese Futurekeys sind im Moment völlig unbrauchbar, da kein Schlüssel mehr existiert, mit dem man sie entschlüsseln könnte. Sie können nur zu einer ganz bestimmten Zeit, die beispielsweise 0,5 – 2,5 Sekunden in der Zukunft liegt, mit dem dann gegebenen System-Zeitwert (oder Timer-Alarmregister-Wert) in den ursprünglichen Zeitschlüssel und Masterkey rückgewandelt werden. (Pos. 4, Fig. 3 und Pos. 6-7, Fig. 1)
Erst dann können mit diesen Ausgangsschlüsseln wieder Chiffrierungs- und Dechiffrierungsaufgaben vorgenommen werden. In der Zeit bis dahin werden anstehende Dechiffrierungs- und Chiffrierungsaufgaben in eine Warteschlange / einen Puffer eingestellt. Sie werden dann (zur Zeit X) von einem Interruptprogramm (Timer-Programm) abgearbeitet das per Timer(-Interrupt) aufgerufen wird. Der Timer wurde bei Erzeugung der Futurekeys mit gesetzt und darf nach dem Setzen weder verändert noch ausgelesen werden können, bis er abgelaufen ist bzw. auslöst.

   Am Ende des Timer-Programms werden die Futurekeys wieder neu erzeugt, der Timer gesetzt und Zeitschlüssel und Masterkey gelöscht. (Pos. 3, Fig. 3)
   Zur Erzeugung eines sicheren Futurekey wird der jeweilige Schlüssel (Masterkey / Zeitschlüssel) mit einem mind. 128-Bit-Schlüssel (dem Zeitcode) verschlüsselt der aus einer anwendungsspezifischen Funktion mit evtl. wechselnden / per Zufallszahl bestimmten Parametern erzeugt wird.
   Das Timer-Programm läuft völlig exklusiv im System sodass während dessen die Ausgangsschlüssel (entschlüsselten Futurekeys) vorhanden sind, niemand darauf zugreifen kann. Dennoch werden sie sehr gut versteckt, so wie auch die Futurekeys.

J. FEHLERKORREKTUR BEIM START DES TIMER-PROGRAMMS

Die Berechnung des Zeitcode zu Anfang des Timer-Programms erfordert an sich dass der vom System abgefragte Systemzeitwert exakt der Zeit X entspricht. Und zwar auf die kleinste Einheit genau die von der Systemzeit verarbeitet wird. Also z.B. 1 Mikrosekunde. Dies ist aber nicht immer sicher zu stellen. Zum einen kann es einige Takte dauern bis der vom Timer ausgelöste Interrupt auch tatsächlich ausgeführt wird, zum anderen vergeht bis dahin aufgrund des technischen Ablaufs ein klein wenig Zeit. Auch die ersten Befehle des Timer-Programms benötigen Zeit. Dies dürfte sich zwar aus heutiger Sicht im Bereich von zwei- maximal dreistelligen Nanosekunden bewegen doch auch in dieser kurzen Zeit kann die Systemzeit auf die nächste kleinste Einheit wechseln.
Dies muss durch eine entsprechende Fehlerkorrektur aufgefangen werden. Hierfür gibt es zwei Möglichkeiten je nachdem ob mit der Timer-Hardwareerweiterung gearbeitet wird oder nicht.

1. Ohne Timer-Hardwareerweiterung - Prüfziffern-Hash:

In diesem Fall wird von den Ausgangsschlüsseln je ein z.B. 8 oder 16 Bit breiter Prüfziffern-Hash (versteckt) gespeichert. 16 Bit sollte dabei die Maximalgröße sein. Es ist klein genug um einen Angriff auf die Schlüssel nicht erheblich zu vereinfachen, reicht aber aus damit das Timer-Programm sicher feststellen kann ob die korrekten Ausgangsschlüssel berechnet wurden. Falls nicht, berechnet es diese nochmals neu mit einem um eine kleinste Zeiteinheit verringerten Systemzeitwert. Dies wird ggf. wiederholt bis der/die Ausgangsschlüssel korrekt berechnet werden konnten.
Empfehlenswerte Größe für den Prüfziffern-Hash sind 8 Bit. Letztlich hängt es von der Leistung des Systems ab und muss entsprechend angepasst werden. Je schneller das benutzte Computersystem arbeitet desto weniger Bit beim Prüfziffern-Hash sind notwendig.
Wenn mehr als 2 Ausgangsschlüssel zu Futurekeys verschlüsselt werden, reicht es dennoch von maximal 2 einen Prüfziffern-Hash zu bilden um zu testen ob die korrekten Ausgangsschlüssel berechnet wurden.

Damit dieser Prüfziffern-Hash es einem Angreifer nicht deutlich leichter macht, die entschlüsselten Ausgangsschlüssel, welche während ihrer Verwendung z.B. in der Zufallsmatrix versteckt sind, zu finden, muss darauf geachtet werden, dass beim Erstellen der Zufallsmatrix mit Zufallszahlen, diese so manipuliert werden, dass ein bestimmter Anteil der aus den Teilgrößen bildbarer Kombinationen der selben Länge wie die versteckten Ausgangsschlüssel,

die selbe Hash-Prüfsumme ergeben müssen wie die echten Ausgangsschlüssel. Dieser Anteil sollte bei ca. 75 - 90% liegen, wenn ein Ausgangsschlüssel in der Matrix versteck wird und entsprechend anteilig wenn es mehrere sind. Also 37,5 - 45% bei 2 Ausgangsschlüssel, 25 – 30% bei 3 Ausgangsschlüssel usw. Ein Beispiel: Ein 128-Bit-Ausgangsschlüssel soll in 8 Teilen á 16 Bit aufgeteilt in der Matrix versteckt werden. Diese 8 Teile werden irgendwo in einer Matrix aus beispielsweise 1 Million 16-Bit-Zufallswerten gespeichert (siehe Beschreibung Zufallsmatrix). Nun müssen 75 - 90% aller möglichen 128-Bit-Kombinationen aus irgendwelchen 16-Bit-Speicherwerten der Matrix denselben Prüfziffern-Hash ergeben wie der darin versteckte Ausgangsschlüssel. Sollen mehrere Ausgangsschlüssel in der Matrix versteckt werden, braucht diese Methode nur für Ausgangsschlüssel angewendet werden, von denen ein Prüfziffern-Hash gespeichert wurde., wobei die o.g. Anteiligkeit gleich bleibt (also z.B. 25-30% bei 3 Schlüsseln). Ohne diese Methode bräuchte ein Angreifer nur all die vergleichsweise wenigeren Kombinationen durch probieren die dem Prüfziffern-Hash entsprechen und würde somit einige Zeit einsparen.

An sich wäre es ebenfalls praktikabel vom Zeitcode einen solchen Prüfziffern-Hash zu bilden, doch die beschriebene Methode bietet vor allem bei der Verwendung von mehreren Ausgangsschlüsseln mehr Sicherheit. Allenfalls könnte eine Kombination von einem etwa 4-Bit Prüfziffern-Hash für Zeitcode und einen ebenso kleinen für einen Ausgangsschlüssel in Frage kommen.

2. Mit Timer-Hardwareerweiterung – Alarm-Register auslesen:

Der Timer der Timer-Hardwareerweiterung hat einige besondere Funktionalitäten.
Handelt es sich um einen sog. Countdown-Timer der von einer gesetzten Zeit an auf 0 zurück läuft, dann muss der Timer nach seinem Auslösen weiter laufen (quasi ins Minus) bis er gelesen wird, so dass das Timer-Programm die erfolgte Zeitverzögerung durch Auslesen dieser Nachlaufzeit abfragen und ausgleichen kann.
Ist es hingegen ein Timer in dem eine beliebige Alarmzeit programmiert werden kann dann wird für diese Alarmzeit bzw. das Timer-Alarmzeit-Register ohnehin der Wert der echten Zeit X verwendet – also dem Wert aus dem der Zeitcode errechnet werden kann, dann bleibt dieser Wert so lange im Alarmzeitregister des Timers erhalten bis er einmal ausgelesen wurde. So braucht das Timer-Programm nur das Alarmzeit-Register (das vor dem Erreichen der Alarm-Zeit (Auslösen) nicht auslesbar ist) lesen und diesen Wert als Basis zur Berechnung des Zeitcodes zu nehmen. Eine eigentliche Fehlerkorrektur ist dann nicht notwendig. Es muss in diesem Fall auch kein Prüfziffern-Hash verwendet werden. Dies ist die vom Erfinder präferierte Variante. Zu

beachten ist auch die Erweiterung wie in Kapitel O und O.1 (Hardwaremäßige Zufallszahlerzeugung) beschrieben sowie die Ausführungen in Kapitel D.

K. VERFAHRENSMECHANISMEN GEGEN ANGRIFFSSZENARIEN

1. Zufallszahlen

An mehreren Stellen des Verfahrens werden Zufallszahlen eingesetzt. Das Problem: Ein Computer kennt eigentlich keinen Zufall. Zufallszahlen werden aus verschiedenen Parametern, wobei die Systemzeit i.d.R. die Hauptrolle spielt, berechnet. Damit ist eine Vorhersage bzw. Manipulation nie ganz auszuschließen.

Wo Zufallszahlen für dieses Verfahren verwendet werden, sollten sich diese daher grundsätzlich aus mehreren Quellen bedienen und daraus eine Kombination bilden. Softwaretechnisch werden z.B. die letzten Bewegungen der Maus gerne mit einbezogen da die Handbewegung eines Menschen nie 100-ig exakt oder gleich ist. Zusätzlich könnte man einen online bezogenen Wert mit einbeziehen. Dabei ist darauf zu achten dass egal welche Zahlen hier heran gezogen werden diese keine Manipulationen bewirken können. Ein Überlauf, Null-Produkt, Maximierung, Minimierung oder Division-By-Zero muss ausgeschlossen werden.

In jedem Fall muss der hier verwendete Zufallszahlengenerator ein nicht-deterministischer kryptografisch sicherer Zufallszahlengenerator hoher Güte sein der außerdem nicht von außen beeinflusst werden kann. Dies lässt sich in der Praxis sicher und effektiv nur mit Hardwareunterstützung realisieren. Hierfür gibt es spezielle Hardware und die mehrfach erwähnte Timer-Hardwareerweiterung ist dafür vorgesehen, auch eine solche hardwarebasierte Zufallszahlerzeugung mit anzubieten. Siehe hierzu Kapitel O.1 (Hardwaremäßige Zufallszahlerzeugung).

2. Geringe Timer-Auflösung und Schlüsselbreite

Wird ohne Timer-Hardwareerweiterung gearbeitet und hat der System-Timer nur eine Auflösung von 1 Mikrosekunde oder höher, dann könnte sich daraus zunächst ein Risiko für einen Brute-Force-Angriff ergeben, da sich bei einem 2 Sekunden Zeitfenster für Zeit X und 1 Mikrosekunde Auflösung des Timers nur 2 Millionen Möglichkeiten ergeben. D.h. der Zeitcode könnte relativ

schnell über das durchprobieren von 2 Mio. verschiedenen Zeitangaben gefunden werden. Vorausgesetzt dem Angreifer ist die anwendungsindividuelle Funktion zur Berechnung des Zeitcodes bekannt.

a) Komplexe Berechnung des Zeitcodes

Um dies zu vereiteln muss die Berechnung eines Zeitcodes mit Hilfe einer anwendungsindividuellen Funktion sehr komplex sein, u.U. mit komplexen Zahlen und hohen (z.B. 2000-stelligen) Potenzen operieren so dass die Berechnung entsprechend lange dauert (Thema Radizieren, Faktorisierung, diskreter Logarithmus, Primfaktorzerlegung).

Ein kompletter Test für eine hypothetisch angenommene Zeit X würde folgende Arbeitsschritte beinhalten:

- Zeit X → Zeitcode (für das reguläre System nur 1-mal pro Timer-Phase durchzuführen)
- Futurekeys → Ausgangsschlüssel (2 mal)
- Dechiffrierung einiger Daten um zu sehen ob Schlüssel funktionieren

Der Rechenaufwand für einen solchen kompletten Durchgang sollte ca. 0.3 Sekunden benötigen, wobei ca. 99% des Zeitaufwands auf die Berechnung des Zeitcodes entfällt, ansonsten wäre das System unbrauchbar. Geht man davon aus dass ein womöglich vom Angreifer eingesetzter Super-Computer zu dem die Berechnungen übertragen würden ca. 10.000-mal schneller ist, so würde dieser nur 0,03 Millisekunden benötigen. (Anm.: Der aktuell 10t-schnelllste Computer der Welt ist theoretisch etwa 30.000-mal schneller als ein Top-PC) So könnte ein Angreifer in 1 Minute alle 2 Millionen Möglichkeiten durch probiert haben.
Da der Zeitcode spätestens alle 3 Sekunden erneuert wird, geht von diesem Angriffsszenario keine erhebliche Gefahr aus, aber mit viel Glück könnte ein Angreifer irgendwann in diesen 3 Sekunden die richtigen Schlüssel finden. Handelt es sich um einen reinen Offline-Angriff, d.h. würde man versuchen den Zeitcode direkt auf dem infizierten Rechner heraus zu finden, wäre dies aufgrund der geringeren Rechenleistung jedoch völlig chancenlos. Dies ist jedoch nur aufgrund der oben erwähnten Komplexität der Fall. Wäre dem nicht so, könnte ein Verbund-Angriff (Schadprogramm auf Server schickt erbeutete Daten und Futurekeys online zu einem Supercomputer im Bereich der Top 50) in 0,05 Mikrosekunden die richtigen Schlüssel finden.

Wichtig ist auch, darauf zu achten, dass durch die Berechnungszeit keine Rückschlüsse auf die Ausgangwerte gezogen werden können.
Die o.g. Komplexität in der Berechnung des Zeitcodes muss daher auch stets an die aktuellen Leistungen von handelsüblichen Top-PC's bzw. Supercomputern angepasst werden. Dennoch ist

dieses Risiko nicht zu vernachlässigen, weshalb für eine echte Unknackbarkeit nachfolgende Lösungen notwendig sind.

Würde der Angreifer jedoch einige verschlüsselte Daten zum Testen und sämtliche Futurekeys erbeuten (also zu einem externen (Super-)Computer übertragen) dann wäre genug Zeit die Ausgangsschlüssel zu finden. Nämlich so lange, wie der Zeitschlüssel gültig ist. Das dürfte im Schnitt eine Stunde sein.

Zusätzlich besteht immer die Gefahr, dass es einem Angreifer gelingt die verschlüsselte Datenbank inklusive aller Futurekeys zu erbeuten und die Funktion kennt mit der aus der Systemzeit bzw. Zeit X der Zeitcode generiert wird.

Auch deswegen sind die nachfolgenden Lösungen und die im Unterpunkt 4 (Technischer Fortschritt) entscheidend.

b) 128 Bit frei wählbares Timer-Register

Als Timer wird eine zertifizierte Hardwareerweiterung genutzt (Anspruch 2), welche einen Timer mit einer um den Faktor 1.000.000 höhere Auflösung haben könnte. Somit erhöht sich die Zeit für einen Angriff wie oben beschrieben schon mal auf knapp 2 Jahre und die Chance auf einen Glücks-Treffer wird verschwindend gering.

Diese Timer-Hardwareerweiterung kann zusätzliche Dienste zur Verfügung stellen. Dazu zählen u.a. eigenständige Verschlüsselungen, Schlüsselverwaltung, Zufallszahlerzeugung und deren Speicherung. Vor allem aber kann dieser Timer auf eine beliebige Zeit als Startzeit und Alarmzeit, mit jeweils mindestens 128-Bit-Registern, gestellt werden. Damit ist die Zeit X, welche zwar relativ immer noch ca. 2 Sekunden in der Zukunft liegt im Timer jedoch mit einem Zufallswert (als Zeit) verschiebbar, wodurch sicher gestellt wird, dass die von einem Angreifer durch zu probierenden Werte deutlich mehr sind als die ca. 2 Sekunden in der niedrigsten Zeitauflösung. Ein Beispiel: Ein Timer hat 64 Bit in dem üblicherweise eine Zeitangabe von Datum und Uhrzeit gespeichert wird. Per Zufallszahl wird diese Zeit nun auf einen beliebigen Wert gesetzt. Eine Art Pseudo-Zeit, die jedoch - im Gegensatz zu dem Standard-System-Timer - nicht abgerufen werden kann. Nun wird die Zeit X abzüglich der aktuellen Zeit (also nur noch die reine Zeitspanne von z.B. 2,3 Sekunden) zu dieser zufällig gewählten Timer-Zeit hinzu addiert und als Alarmzeit programmiert (die für einen Angreifer ebenfalls nicht auslesbar ist). Diese neue Alarmzeit ist nun eine völlig variable 64-Bit Basis für den Zeitcode welcher vom Timer-Programm später rekonstruiert werden kann da dieses die dann vorliegende Timer-Alarmzeit nur auszulesen braucht um den Zeitcode erneut zu berechnen. Mit 64 Bit verschiedenen Möglichkeiten müsste ein Angreifer schon etwa 2 Millionen Jahre investieren.

Gut, - man kann davon ausgehen dass er nicht alle Möglichkeiten durch probieren muss bis die wahren Ausgangsschlüssel gefunden sind, aber wären diese im ersten Zehntel enthalten bleiben immer noch 200.000 Jahre.

Bedenkt man jedoch, dass sich die Leistung von Super-Computern ca. alle 10 Jahre vertausendfacht, so wird deutlich, dass es bis zu einem ernsthaften Gefahrenpotential nicht allzu lange dauern würde, wenn nicht die o.g. Komplexität der Zeitcode-Berechnung stetig angepasst wird. Aus diesem Grund wird - wie oben erwähnt - ohnehin ein mindestens 128-Bit-Timer empfohlen (siehe nachfolgender Unterpunkt 4 „Technischer Fortschritt“). Ist dies gegeben, kann aus Zeitersparnis-Gründen die oben dargestellte Komplexität der Berechnung von Zeitcodes weggelassen oder verringert werden und das Verfahren ist gut 90 Jahre auf der sicheren Seite. Jede zusätzlichen 64 Bit bringen weitere knapp 65 Jahre Sicherheit.

3. Schnittstellen-Kommunikation

Bei jeder Kommunikation besteht das Risiko des Mithörens oder der Manipulation. Aus diesem Grund muss für ein durchgängig sicheres System dieses Risiko ausgeschaltet werden.

Bekannt ist die Kommunikation zwischen Web-Server und Web-Browser. Hier wurde mit TLS/HTTPS, vor allem mit seinen Erweiterungen (HSTS, Zufallswerterweiterung gegen DNS-Spoofing, Station-to-Station-Protokoll, HTTP Public Key Pinning), ein guter Sicherheitsstandard geschaffen. Um diesen zu erweitern wird bei allen Kommunikationen in Verbindung mit dem vorliegenden Verfahren (zwischen Programmen und zwischen Hardware), was insbesondere auch für die Kommunikation mit dem Aufgabenpuffer des Timer-Programms gilt, folgendes Verfahren angewandt, was als Erweiterung des o.g. Sicherheitsstandards (TLS etc.) zu sehen ist:

- Client sendet Kommunikations-Anfrage an Server
- Server sendet einen öffentlichen Schlüssel (OS1) für asymmetrische Verschlüsselung nebst Puplic-Key-Zertifikat an den Client.
- Client überprüft Zertifikat bei vertrauenswürdiger Zertifizierungsstelle.
- Client sendet ggf. eigenes Zertifikat.
- Client erzeugt eine z.B. 128-Bit Zufallszahl als symmetrischen Schlüssel (SS) die er – verschlüsselt mit OS1 - an den Server zurück sendet.
- Server entschlüsselt den SS mit seinem privaten asymmetrischen Schlüssel.
- Server generiert Grundwerte eines Diffie-Hellman-Merkle-Schlüsselaustausch (DHM), Primzahl und Generator, sowie seinen öffentlichen DHM-Schlüssel (DHM1) und sendet dies - verschlüsselt mit SS - an den Client.
- Client entschlüsselt DHM-Grundwerte und -Schlüssel DHM1 mit SS.

- Client berechnet mit den erhaltenen DHM-Grundwerten seinen öffentlichen DHM-Schlüssel (DHM2) und sendet diesen - wieder mit OS1 verschlüsselt - an den Server.
- Server entschlüsselt den DHM2 mit seinem privaten asymmetrischen Schlüssel.

BEIDE haben nun ihre öffentlichen DHM-Schlüssel ausgetauscht und können damit den geheimen DHM-Schlüssel errechnen und die eigentliche Kommunikation beginnen. Damit kann nun auch ein neuer symmetrischer Schlüssel ausgetauscht bzw. aus den DHM-Schlüssel generiert werden, über den dann die gesamte Kommunikation verschlüsselt wird, da die DHM-Grundwerte hier aber (unterschiedlich) verschlüsselt übertragen wurden, wurde die Sicherheit für das DHM deutlich gesteigert (jedoch bei weitem nicht so wie im Kapitel G.3).

Dies ist eine erweiterte Variante des TLS-Protokolls, welches normalerweise jedoch - nach heutigem Wissensstand - ausreichend sein sollte. Insbesondere das Station-to-Station-Protokoll mit zusätzlichen digitalen Signaturen und Message-Authentication-Codes dürfte aktuell ausreichend sein.

Die zu erwartende rasante Entwicklung der Rechenleistung könnte jedoch die weitere Stufe sinnvoll machen. Bei TLS wird bezüglich asymmetrischer Verschlüsselung und DHM entweder oder angewandt. Hier ist es ein und. Zudem werden unterschiedliche Verschlüsselung benutzt. Entscheidender Vorteil dabei: Selbst wenn ein Angreifer die gesamte Kommunikation aufzeichnen kann und irgendwann genug Rechenleistung zur Verfügung hat, die asymmetrische Verschlüsselung zu brechen, erhält er damit eben nicht den Session-Key und kann die Kommunikation nachträglich nicht entschlüsseln. Stattdessen erhält er nur die Grundwerte und Schlüssel des DHM-Protokolls und müsste nun erst noch dieses brechen, was bei entsprechenden Größen der Grundwerte (> 2000 Bit), lange als sicher gelten darf. Dieser Umstand erschwert das Brechen des asymmetrischen Schlüssels nicht unerheblich.

Ein weiterer Vorteil: Die Grundwerte (oder zumindest einer davon) des DHM-Protokolls könnten von der Zertifikatstelle vergeben werden. So werden diese an beide Kommunikations-Partner gegeben ohne dass sie Teil der eigentlichen Kommunikation sind. Das wiederum bedeutet sie sind vom Angreifer nicht zu erbeuten selbst wenn er die symmetrische Verschlüsselung gebrochen hat, bzw. es müsste dann erst die Verschlüsselung des Zertifikats gebrochen werden. Da es bereits bekannt ist dass DHM bei Bekanntwerden der DHM-Grundwerte (oder einer bestimmten Untermenge) und Schlüsselgrößen unter 512 Bit, durchaus in kurzer Zeit zu brechen ist, sollte dieses Verfahren nie so angewandt werden, dass diese Grundwerte mitgehört werden können. D.h. diese Grundwerte dürfen nicht unverschlüsselt übertragen werden. Die Empfehlungen des BSI zu der Länge dieser Grundwerte müssen

berücksichtigt werden. Ebenso sollte das DHM-Verfahren in kurzem Abstand erneuert werden und zwar innerhalb des bisherigen, sodass die Notwendigkeit einer stetigen Neu-Brechung besteht.

Damit das sichere Datenschutzverfahren der vorliegenden Erfindung nicht anderweitig ausgehebelt wird sollten zur Kommunikation stets die aktuell sichersten Kommunikations-Protokolle und -Verfahren zur Anwendung kommen.
Bei der Übermittlung von Passwörtern wird dieses Verfahren nochmals erweitert. Siehe G.3.

Für die Übertragungen zum Aufgabenpuffer dürfte aufgrund der kurzen Lebensdauer desselben eine langschlüsselige asymmetrische Verschlüsselung reichen. D.h. das Timer-Programm kreiert bei jedem Durchlauf einen privaten Schlüssel der bis zu seinem Gebrauch mit einem veränderten Masterkey und Zeitschlüssel verschlüsselt wird und einen öffentlichen Schlüssel der an alle Kommunikationspartner gegeben wird. Da es zu Überschneidungen kommen kann wird jeweils der letzte private Schlüssel ebenfalls aufgehoben. Eine kurze (8 oder 16 Bit) Prüfziffer bei jedem Puffer-Eintrag hilft sicher zu stellen dass der richtige private Schlüssel angewandt wird.

Das Timer-Programm wird außerdem verschiedene Algorithmen nutzen, um zu erkennen, ob die Einträge im Aufgabenpuffer ggf. von einem Angreifer stammen und entsprechend reagieren. Da Anmeldedaten generell nie mehr entschlüsselt werden, könnte ein Angreifer maximal 4-fach verschlüsselte Werte erlangen zu dessen Entschlüsselung die Lebensdauer des Weltalls, erst nach weiteren 300 Jahren technischer Fortentwicklung, ausreicht.

4. Technischer Fortschritt

Heute kratzen bereits die ersten Supercomputer an der 100-PetaFlops-Marke. Irgendwann sind womöglich – auch wenn es heute unmöglich erscheint – selbst solche ausgefeilt mehrfach verschachtelten Verschlüsselungen knackbar, wie sie die vorliegende Erfindung präsentiert. Jemand der durch einen sehr geschickten Angriff die komplette Datenbank und die Futurekeys erbeutet, könnte diese womöglich in X Jahren knacken.
Ein Beispiel: Der 10t-schnellste Computer (10 PetaFlops) wird eingesetzt um einen 64-Bit-Schlüssel zu knacken. Es wird kein Algorithmus verwendet sondern schlicht alle Möglichkeiten durchprobiert, wobei pro Durchgang 50 Flops veranschlagt werden. Er wäre in maximal 25 Stunden durch. Und, - der richtige Schlüssel liegt - statistisch gesehen - höchstwahrscheinlich

nicht erst ganz am Ende der Testreihe.
Dies macht deutlich, dass 64 Bit heutzutage keine nennenswerte Sicherheit mehr darstellen.

a) 128-Bit-Timer-Register

Aus diesem Gesichtspunkt ist es entscheidend dass der Zeitcode mindestens 128 Bit verschiedene Möglichkeiten enthält. Das bedeutet, seine Werte müssen auf einen 128-Bit-Timer basieren. Oder anders ausgedrückt: Wer den Zeitcode brechen will, muss mindestens 2^128 Möglichkeiten durchspielen. Die anwendungsindividuelle Funktion erhöht die Sicherheit zusätzlich auf z.B. 192 Bit, aber wenn einem Angreifer diese Funktion bekannt geworden ist, dann reden wir definitiv von 2^128 Möglichkeiten.
Ein Beispiel: Ein Angreifer erbeutet die gesamte Datenbank nebst Futurekeys und Zeitcode-Funktion. Die Gesamt-Verschlüsselung der Datenbank zu brechen ist noch in hunderten Jahren unrealistisch. Deshalb würde man sich auf die Brechung des Zeitcode konzentrieren (was natürlich nur Sinn macht, wenn man die Futurekeys erbeutet hat). Die Funktion liegt vor (wurde ebenfalls entwendet) weshalb die Angreifer maximal 2^128 Durchläufe bräuchten. Die Lösung liegt aber nicht erst am Ende sondern angenommenerweise ungünstig im ersten Promille. Somit bleiben noch 2^118 Durchläufe, wobei ein Durchlauf 50 Flops benötigt (keine Verkomplizierung). Es würde 52 Billionen Jahre dauern.

Allerdings könnten in 30 Jahren SuperComputer realistisch 1e9 mal so schnell sein und dann wären es „nur“ noch 52.687 Jahre. Doch was ist, wenn in 20 Jahren tatsächlich Quantencomputer eingesetzt werden können? Bei einem durchaus nicht völlig unrealistischen weiteren Entwicklungssprung von 1e6 ist der Schlüssel in 19 Tagen geknackt.
Dies macht deutlich warum eine Bitbreite von 128 in allen Schlüsselrelevanten Bereichen als Minimum angesehen werden sollte und der Zeitcode auf mindestens 192 Bit empfohlen wird.
Weiterhin wäre bei Daten, die auch in 30 oder 50 Jahren noch sicher sein sollen, ein 192 oder 256 Bit Timer durchaus anzuraten. 64 zusätzliche Bit schaffen eine Verbesserung um den Faktor 18 Trillionen. Das reicht dann wieder 60 bis 70 Jahre. Eine Verkomplizierung der Zeitcode-Funktion kann bis zu einem Faktor von 1000 Sinn machen und bietet auf alle Fälle etwas mehr Sicherheit (ca. 10 Jahre).
Die Variante per Zufallszahl aus einem 128-Bit-Timer echte 2^192 notwendige Durchläufe heraus zu holen ist in Kapitel O.1. (Hardwaremäßige Zufallszahlerzeugung) beschrieben.
Noch besser ist es jedoch, wenn es einfach - wie nachfolgend ausgeführt - nicht möglich ist, die Futurekeys komplett zu erbeuten.

b) Futurekeys verschlüsseln und aufgeteilt verstecken

Dazu wird der Futurekey des Masterkey's (MK-Futurekey) in einem oder mehreren ausreichend großen Speichermedien, außer dem Arbeitsspeicher des Systems, versteckt und zwar so dass es unmöglich ist einen gültigen kompletten Futurekey durch Durchsuchen oder Downloaden und nachträgliches Durchsuchen zu erlangen. Es darf nicht möglich sein, das dafür verwendete Speichermedium / Speicherbereich komplett zu erbeuten in weniger als z.B. dem 10-fachen der Zeit in der der versteckte Schlüssel, oder der Zeitschlüssel mit dem er ggf. verschlüsselt wurde, längstens Gültigkeit hat. Dies muss hardwareseitig durch das richtige Verhältnis von Lesegeschwindigkeit des Speichermediums, dessen Größe und der maximalen Zeitspanne bis zur (fast) regelmäßigen Erneuerung des versteckten Schlüssels bzw. Zeitschlüssel sowie einer Aufteilung des zu versteckenden Schlüssels auf das gesamte Speichermedium sichergestellt sein.

Dieses wichtige Merkmal dieses Verfahrens entspricht Anspruch 7...

L. BESCHREIBUNG ANSPRUCH 7: (HDD-MATRIX – VERSTECK FUTUREKEYS)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
Daten, insbesondere aber kryptografische Schlüssel mit einem, in bestimmten Zeitabständen zu erneuernden, Schlüssel (Zeitschlüssel) verschlüsselt und dann in mehreren Teilen aufgesplittet auf einem (mit Zufallswerten gefüllten) Speichermedium verteilt gespeichert werden, wobei der maximale Zeitabstand zur Erneuerung des Zeitschlüssels und/oder Schlüssels und/oder deren Neu-Verschlüsselung, die Größe des Speichermediums und dessen Lesegeschwindigkeit so im Verhältnis stehen, dass gilt: Größe in MByte >= Lesegeschwindigkeit in MByte/Sek. * Vielfaches * maximaler Zeitabstand zur Erneuerung/Neuverschlüsselung des Schlüssels in Sekunden.

Die bestimmten **Zeitabstände** werden im Bereich des Anspruch 4 zum Thema Zeitschlüssel ausführlich erklärt. Im Wesentlichen dürfte die Größenordnung je nach Performance des Systems zwischen 1 Stunde und 1 Tag liegen, wobei die letztlich festgelegte Größe in der Praxis eine gewisse Variabilität haben muss um nicht vorhersehbar zu sein.
Mit Schlüssel (Zeitschlüssel) sind hier aber nicht nur Zeitschlüssel im Sinne des Anspruchs 4 gemeint sondern auch alle sonstigen Schlüssel und insbesondere auch Futurekeys, deren Gültigkeit im Schnitt nur einige Sekunden beträgt.

1. Gleichmäßige Verteilung

Ein **Speichermedium** wird vorab einmal komplett mit Zufallszahlen beschrieben. Die Größenordnung der Zufallszahlen muss derselben entsprechen wie ein Teil des aufgesplitteten (verschlüsselten) (Masterkey-Futurekey-) Schlüssels der versteckt werden soll. Das wäre z.B. bei einer Aufsplittung eines 128-Bit-Schlüssels in 8 Teile je ein 16-Bit-Wert. Es muss möglich sein, auf einzelne Bereiche/Zellen des Speichermediums direkt zugreifen zu können um dann die Teile des aufgesplitteten (verschlüsselten) Schlüssels dort direkt hinein „pflanzen“ (schreiben) zu können. So wie man bei Arbeitsspeicher direkt auf jede beliebige Speicherzelle zugreifen kann.

Die Speicherorte der Teile des aufgesplitteten Schlüssels sollten völlig zufällig, aber gut **verteilt** sein. So muss es eine hohe Wahrscheinlichkeit geben dass sich ein Teil von Y (1/Y des Schlüssel) auch ungefähr in 1/Y des Speichermedium befindet, wobei Überschneidungen auf bis zu 1,8/y des Speichermedium gestattet sind. Das könnte unter der Annahme dass das Speichermedium 1000 MB groß und der Schlüssel in 8 Teile gesplittet wird, so aussehen: Ausgehend davon dass das Speichermedium bereits einmal in 16-Bit-Zellen formatiert und mit Zufallszahlen gefüllt wurde, wird es gedanklich in 8 Bereiche geteilt. Anfangs und Ende wird ein kleiner variabler Puffer von je 12 MB eingebaut. Jedes Achtel des Schlüssels soll also in einem Sektor der Größe 122 MB Platz finden wobei eine gelegentliche Überschreitung der Sektorgrenzen bis zu 180% gewollt ist. Dabei müssen natürlich die Grenzen des Speichermedium beachtet werden sowie dass sich keine Schlüsselteile übereinander oder nebeneinander befinden. Der erste Sektor startet demnach bei 12, der zweite bei 12+122=134 usw. In diesen Sektoren wird nun je ein Achtel des Schlüssels untergebracht, wobei eine gelegentliche Überschneidung gewollt ist. Auch sollte die Reihenfolge der Teile auf dem Speichermedium nicht der tatsächlichen entsprechen. Die Formel zur Bestimmung der Speicherorte könnte so aussehen:
Zelle = Mittelwert von 2 Zufallszahlen aus 2,6 minus 0,8 plus Startnummer * 122 + 12, Mindestens 0, Maximal 1000; jeweils multipliziert mit 500.000 ergibt dann die 16-Bit-Adresse
Es wäre auch eine zufallsbestimmte Speicherortbestimmung über den gesamten Datenträger denkbar. Es müsste dann aber sichergestellt sein, dass eine zu ungleichmäßige Verteilung nicht stattfindet.

2. Besonders langsames Speichermedium

Ein „**Vielfaches**“ bestimmt den wievielten Teil des gesamten Speichermedium man innerhalb einer maximalen Zeitspanne der Gültigkeit des versteckten Schlüssels lesen bzw. herunterladen

kann. Als Minimum wird 3 angesehen. Wird als Vielfaches z.B. der empfohlene Wert 10 eingesetzt kann – egal wer und von wo – nur 1/10 des Speichermedium gelesen werden bis der versteckte Schlüssel schon wieder erneuert wird. Somit hat man nie die Chance auf mehr als 1/10 eines korrekten Schlüssels. Während der Angreifer weiter liest, werden bereits die anderen 9/10 mit Teilen von neu verschlüsselten Schlüsseln versehen und die alten sicher gelöscht. Am Ende hätte der Eindringling lediglich 10 Schnipsel von jeweils unterschiedlich verschlüsselten Schlüsseln.

Damit auch keine Teile alter (verschlüsselter) Futurekeys in einem evtl. erbeuteten Speichermedium zu finden sind, werden diese, sobald sie nicht mehr benötigt werden sicher gelöscht und mit Zufallswerten gefüllt.

Geht man beispielsweise von einem Zeitrhythmus für die Erneuerung des Zeitschlüssels von 1 Stunde und von einer bestenfalls (in dem Fall schlimmstenfalls) durchschnittlichen Lese/-Downloadrate von 100 Mbit/Sek. aus, so muss der Datenträger, in dem der mit dem Zeitschlüssel verschlüsselte MK-Futurekey versteckt wird mindestens 450 GB groß sein.

Größe in MB >= Geschw. in MByte/Sek. * (10 * Zeitschl.Ern.Zeit in Sekunden)

Geschw. in MByte/Sek. <= Größe in MB / (10 * Zeitschl.Ern.Zeit in Sekunden)

Mit „Geschw." ist die Lese- bzw. Übertragungsgeschwindigkeit vom Speichermedium gemeint und unter „Zeitschl.Ern.Zeit" ist die maximale Zeitspanne bis der versteckte Schlüssel immer wieder erneuert wird zu verstehen.

Eigentlich bemüht man sich die Lesegeschwindigkeit bei Speichermedien ständig weiter auszubauen. Für diesen Bereich des vorliegenden Datenschutz-Verfahrens wird die Lesegeschwindigkeit bewusst begrenzt. Dies muss hardwareseitig fixiert und unveränderlich sein. Die Begrenzung sollte so ausfallen, dass sie gerade noch keine spürbare Verzögerung der Abarbeitung der Chiffrierungsarbeit (des Timer-Programms) bewirkt. Ist das Speichermedium groß genug und die Zeit bis zur Erneuerung des versteckten Schlüssels sehr kurz dann kann u.U. (kein Cache z.B.) auch auf eine Begrenzung der Lesegeschwindigkeit verzichtet werden.
Ein Futurekey ist grundsätzlich nur etwa 3 Sekunden gültig bis er entnommen, entschlüsselt, verwendet und wieder neu (anders) verschlüsselt wird. Geht man von einer Lesegeschwindigkeit von 1 GByte/Sek. aus, was derzeit eine gute Geschwindigkeit einer modernen SSD ist, so müsste der Speicher gerade mal 30 GByte groß sein.

Vom Grunde her kann man diese etwa 3 Sekunden bis zur Erneuerung der Futurekeys als Zeitspanne ansetzen und ist ausreichend geschützt. Der Aufwand dafür ist gering. Eine weitere

Verschlüsselung der Futurekeys mit z.B. dem Zeitschlüssel ist an sich nicht notwendig. Es muss dann aber auch sichergestellt sein dass es selbst bei einem System-Neustart keine Zeitspanne gibt, in der ein gültiger Schlüssel länger als 6 Sekunden im Speicher verbringt. Andernfalls ist auf die längere Zeitspanne abzustellen.

Zwar könnte man sich sogar auf die noch langsamere Download-Rate berufen, doch könnte ein Angreifer dann zunächst den Matrix-Speicherbereich auf einen anderen Bereich/Datenträger kopieren und diesen dann in aller Ruhe downloaden.
Auch muss darauf geachtet werden dass der Matrix-Speicherbereich nicht in irgendeiner Weise zu sperren ist, denn auch dadurch könnte der Angreifer verhindern dass z.B. 7/8 des Schlüssels im Matrix-Speicherbereichs bereits wieder erneuert wurden bevor er erbeutet werden konnte.

Der Masterkey-Futurekey wird also (ggf. mit dem Zeitschlüssel verschlüsselt) einigermaßen gleichmäßig aufgesplittet in einer Matrix gleichgroßer zufällig gewählter Schlüsselteile auf einem Speichermedium außerhalb des Arbeitsspeichers versteckt. Wo, wird per Zufall bestimmt und in Zeigern vermerkt, welche (ebenfalls) mit dem Zeitschlüssel verschlüsselt werden. Dies wird vom Timer-Programm und somit etwa alle 3 Sekunden ausgeführt. Siehe Fig. 5 und 6.
Die Zeiger sind für die kurze Zeit mit der Zeitschlüssel-Verschlüsselung ausreichend geschützt, könnten aber auch noch versteckt werden. Siehe hierzu weiter unten die Ausführungen zum Verstecken von Zeigern zum Zeitschlüssel-Futurekey.

Eine weitere Erhöhung der Sicherheit kann erreicht werden indem das Versteck für den Masterkey-Futurekey per Zufallsauswahl auf mehrere, idealerweise typverschiedene, Geräte verteilt wird, so dass nie sicher ist, auf welchem/n Gerät(en) der Futurekey versteckt ist. Wobei damit verschiedene Speichermedien des Systems (nicht der Arbeitsspeicher in dem auch der Zeitschlüssel-Futurekey gespeichert ist) gemeint sind aber auch Speicher anderer Systeme die u.a. per Netzwerk angebunden sind oder sichere (Cloud-)Server im Internet, vorausgesetzt sie garantieren die oben dargestellte Begrenzung der Lesegeschwindigkeit in Relation zu ihrer Größe und der Erneuerungszeit des Zeitschlüssel oder Futurekeys. Auch ein geschützter Speicherbereich auf der Timer-Hardwareerweiterung kommt dafür in Frage. Notwendig wäre dieser Schritt jedoch nur wenn die Gefahr der physischen Entwendung des einzelnen Speichermediums besteht, wogegen sich die Timer-Hardwareerweiterung absichern kann. Die Timer-Hardwareerweiterung hätte allerdings zusätzlich den Vorteil, dass sie Lesezugriffe während der Zeit in der das Timer-Programm nicht aktiv ist generell gar nicht zulässt.
Der klare Nachteil eines Netzwerk-Speichermediums liegt darin, dass der Netzwerkverkehr womöglich abgehört werden könnte, womit das gesamte Versteck sinnlos wäre.

Es spielt dabei an sich keine Rolle ob die Festplatte selbst so langsam ist oder die Verbindung dahin, doch ersteres ist klar zu bevorzugen da es Netzwerk- oder Anbindungs-Manipulationen sinnlos macht. Stellt beispielsweise ein 1-TB-Speichermedium nur 10 Mbit/Sek. Übertragungsgeschwindigkeit zur Verfügung, dann spielt das für den Abruf des Schlüssels seitens des Timer-Programms, das genau weiß wo er sich befindet, keine große Rolle. Es dauert dann eben statt 1,3 Mikrosekunden 13 Mikrosekunden was einen 3 Sekunden-Aufruf nur unwesentlich verlängert. Ein Download dieser 1-TB-Festplatte würde jedoch 9 Tage dauern.

Hat man alles erbeutet außer dem Masterkey dann bedeutet das, dass es für einen Angriff mehr Sinn macht die Brechung der beiden Ausgangsschlüssel Masterkey und Zeitschlüssel (zusammen 256 Bit) anzugehen anstatt Zeitcode und danach Masterkey (zusammen 320 Bit). Selbst in 150 Jahren, wenn Supercomputer die unvorstellbare 1e45-fache Leistung von heute haben, wird es immer noch unmöglich sein, eine so geschützte Datenbank zu knacken, da dafür 2^256 Möglichkeiten durchgerechnet werden müssten. Dafür sind dann immer noch 18 Milliarden Jahre notwendig. Dann sollte man aber spätestens langsam auf 512 Bit umstellen. Das reicht dann weitere 250 Jahre. Natürlich kann niemand sagen, ob die Entwicklung der Rechengeschwindigkeit so rasant weiter geht. Irgendwo sind physikalische Grenzen gesetzt. Es könnte aber womöglich sogar auch noch schneller gehen wenn größere Quantensprünge erreicht werden.

Beim Speichern auf Festplatte(n) wird hier idealerweise der sog. Direktzugriff angewandt um den Futurekey irgendwo abzulegen ohne dass das Filesystem einen Hinweis darauf vermerkt. Generell muss der Speichervorgang so aufgebaut werden, dass keine Spuren hinterlassen werden oder solche müssen gesondert gelöscht werden. Ein Caching sollte deaktiviert sein und generell nicht aktivierbar sein. SSD's sind beispielsweise aufgrund des internen Speichermanagement ungeeignet, außer man kann mit einer modifizierten Firmware ein Pendant zum Direktzugriff der Festplatten erreichen.
Damit das Datenschutz-System den MK-Futurekey wieder finden kann wird der genaue Ort des Verstecks bzw. der Verstecke in Zeigern gesichert welche mit dem Zeitschlüssel verschlüsselt werden. Die Erbeutung dieser wäre belanglos so lange der Zeitschlüssel nicht binnen der seiner Gültigkeit oder gar der Gültigkeit der Futurekeys geknackt werden könnte. Wird aus Performance-Gründen der Zeitschlüssel nur täglich erneuert, können die Zeiger sicherheitshalber zusätzlich mit dem Zeitcode, der sich etwas alle 3 Sekunden ändert, verschlüsselt werden.

3. Zeitliche Brisanz

So ist der MK-Futurekey erst nach erfolgreich geknackten Zeitschlüssel(-Futurekey) zu finden doch bis dahin ist er schon längst woanders, da sich dessen Aufenthaltsort ca. alle 3 Sek. ändert und außerdem ist er nicht mehr gültig. Das komplette Entwenden aller möglichen Aufenthaltsorte des MK-Futurekey scheidet aufgrund der enormen Größe und Vielfalt dieser möglichen Verstecke - vor allem innerhalb der Gültigkeit des Zeitschlüssel oder gar des Futurekeys (Zeit X, beispielsweise 3 Sek.) - aus. Nach der aktuell bevorstehenden Zeit X werden beide Ausgangsschlüssel bereits wieder mit einem neuen Zeitcode zu neuen Futurekeys chiffriert. Es ist also aufgrund der Größe der möglichen Aufenthaltsorte nicht möglich beide Futurekeys zu erbeuten so dass beide mit demselben Schlüssel (Zeitcode) erzeugt wurden. Somit würde es nicht mehr reichen nur den Zeitcode zu knacken. Dies würde nur zu einem der zwei Ausgangsschlüssel, nämlich dem Zeitschlüssel führen, nun müsste aber erst noch der Zeitcode gefunden werden mit dem der jüngere bzw. später erhaltene Masterkey-Futurekey erzeugt wurde. Selbst wenn man alle 0,5 Sekunden einen Speicherdump machen könnte während man alle potentiellen Speicherorte herunter lädt, hätte man aufgrund der langen Downloadzeit für jede Zeit X immer nur einen kleinen Teil des dazu gültigen Masterkey-Futurekey der völlig nutzlos ist.

4. Unabhängigkeit von Rechengeschwindigkeit und technischer Entwicklung

Nur wenn der Zeitschlüssel innerhalb seiner Gültigkeit gefunden werden kann, d.h. der Zeitcode wird geknackt womit aus dem Zeitschlüssel-Futurekey der Zeitschlüssel dechiffriert werden kann, könnte überhaupt über Entschlüsselung der Zeiger die Verstecke der Masterkey-Futurekey-Teile gefunden werden und gezielt erbeutet werden. Da sich aber der Zeitcode alle etwa 3 Sekunden ändert und damit auch die Futurekeys und deren Aufenthaltsort(e), müsste das alles in unter 3 Sekunden ablaufen. Bis ein Supercomputer jedoch 2^192 Möglichkeiten in 3 Sekunden durchspielen kann vergehen noch knapp 134 Jahre, doch das Problem ist eigentlich, dass der Angreifer erst weiß ob er den richtigen Zeitschlüssel hat, wenn er damit dann auch den Masterkey erbeutet um dann mit beiden einige Daten auf Klartext zu testen. Das wiederum bedeutet es kann nicht auf einem externen System allein getestet werden, es müsste für jeden Test ein Download der potentiellen Teile des Masterkey-Futurekey von seinen (besonders langsamen) Versteck-Speichermedien erfolgen weshalb diese Zeit und nicht die Entwicklung der Rechengeschwindigkeit auf ewig der Flaschenhals sein wird. Und wenn jeder Zugriff auf einen potentiellen Masterkey-Futurekey beispielsweise 13 Mikrosekunden dauert, dann reden wir hier von 2,6e42 Jahren bis 2^192 Möglichkeiten durch probiert sind. Und das wird immer so bleiben,

egal wie schnell die Computer dieser Welt noch werden.
Das bedeutet, die Begrenzung der Zugriffszeit eines extern (nicht im Arbeitsspeicher) versteckten Futurekeys hebelt jegliche Rechenpower völlig aus und sorgt nicht nur für die Unmöglichkeit des kompletten Diebstahls um dann in Ruhe nach dem Schlüssel zu suchen, sondern vereitelt auch jeden Brute-Force-Angriff bis in alle Ewigkeit.

Übrigens: Aus einem 450 GB Speichermedium einen in 8 (2 Byte-) Teilen gesplitteten Schlüssel zu finden bedeutet 225e9! : (225e9-8!) : (8!) = 1,6e86 Möglichkeiten und entspricht damit ungefähr einer 286-Bit-Verschlüsselung. Gut, - durch die Sektorierung (gleichmäßige Verteilung) verringern sich die Möglichkeiten, doch dann muss noch die richtige Reihenfolge gefunden werden und schon sind wir wieder bei 1,5e88. Mit einer Verdoppelung der Splitt-Teile könnte gleichfalls die Anzahl der Möglichkeiten verdoppelt werden, was jedoch aufgrund der Unmöglichkeit des Lesens in notwendiger Zeit als unnötig angesehen werden kann.

Idealerweise wird auch der Futurekey des Zeitschlüssels in einem solchen, aber möglichst anderen, Speichermedium auf die gleiche Weise wie der Masterkey-Futurekey versteckt.

Dieser Verfahrensteil insgesamt ist übrigens sogar auch ein wirksamer Schutz gegen einen Prozessor-Dump, falls sich dieser nicht dauerhaft sicher deaktivieren lässt. Zwar ließe sich mit dem Dump der Zeiger mit erbeuten und womöglich auch irgendwann finden und entschlüsseln (= Zeitcode brechen), doch bis dahin ist der Schlüssel auf den er zeigt längst nicht mehr dort vorhanden und das Speichermedium in dem der Schlüssel zu finden gewesen wäre, konnte nicht mit erbeutet werden weil es in der Zeit in der sich der Schlüssel dort aufgehalten hat nicht komplett zu lesen war.

Mann könnte die jeweiligen Zeiger auf die versteckten Futurekeys auch noch verstecken aber es bringt im Gesamten nur einen marginalen Sicherheitsgewinn.

Durch diesen wichtigen Teil des Gesamt-Verfahrens, wird der Angreifer gezwungen, den Zeitcode und damit den Zeitschlüssel innerhalb von ca. 3 Sekunden zu knacken, was selbst ohne Timer-Hardwareerweiterung unmöglich ist und insgesamt ein unknackbaren Verfahren bietet selbst ohne Timer-Hardwareerweiterung, vorausgesetzt der vorhandene Timer lässt sich nicht auslesen. Denn die in Kapitel K.2.a) (Komplexe Berechnung des Zeitcodes) dargestellte Lösung hat nun mal den Nachteil dass sie von der Rechenleistung abhängig ist. Dies ist mit der hier dargestellten Lese-Verzögerung nicht der Fall. Diese gilt auch in 1000 Jahren noch und diese ist unabhängig davon ob der Angreifer einen SuperComputer zur Verfügung hat oder nicht. Wendet

man also die in K.2.a) dargestellte Verzögerung von 0,3 Sekunden durch die Lesegeschwindigkeit an, so würde selbst bei einem niederwertigen Countdown-Timer die Anzahl der (nur 2 Millionen) Möglichkeiten ausreichen um einen Angriff zum reinen Glücksspiel werden zu lassen. Aber, - nach einem Tag und 30.000 Versuchen könnte man statistisch gesehen dieses Glück einmal haben, weshalb es schon zumindest ein Timer mit einer deutlich höheren Auflösung als 1 Mikrosekunde sein sollte oder gleich ein 32-Bit-Alarm-Timer sein sollte. Bei diesen könnte es jedoch unwahrscheinlich sein, dass er unlesbar ist oder so konfiguriert werden könnte.

Ergo: Entweder Countdown-Timer mit Auflösung von 1 Nanosekunde oder kleiner, oder doch gleich eine Timer-Hardwareerweiterung.

M. BESCHREIBUNG ANSPRUCH 8: (FESTSPEICHER - HARDWARE-ERWEITERUNG)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
die sicherheitsrelevanten Programme (u.a. Software zur Chiffrierung und Dechiffrierung / o.g. Timer-Programm) und Daten auf einem nur per manuellem Hardwareeingriff änderbaren/beschreibbaren nichtflüchtigen Speicherchip (z.B. EEPROM auf der o.g. Timer-Hardwareerweiterung) untergebracht sind und entweder sich dieser Speicherchip direkt in einem bestimmten Speicherbereich des Arbeitsspeicher des Systems einblendet oder die o.g. sicherheitsrelevanten Programme und Daten daraus geladen und regelmäßig damit verglichen werden.

Programme im Arbeitsspeicher (RAM) oder auch auf Festplatten/SSD's können manipuliert werden. Dies kann dazu führen, dass sie die besonders geschützten Daten oder Schlüssel preisgeben. Aus diesem Grund muss derartigen Manipulationen vorgebeugt werden. Die dabei sicherste Variante ist ein sog. Festspeicherchip, wie z.B. der ROM (Read-Only-Memory). Dieser hat jedoch den Nachteil dass er nicht updatefähig ist. Daher wird heute auch die Basis-Start-Software von Computersystemen wie z.B. das Bios auf Flashspeicher untergebracht. Der Nachteil: Sie lassen sich rein per Software umprogrammieren. Dies ist jedoch für die vorliegende Erfindung und dem Anspruch der Unknackbarkeit nicht akzeptabel. Aus diesem Grund muss es

einen nichtflüchtigen Speicher (d.h. er verliert seinen Speicherinhalt auch bei ausgeschalteten/stromlosen Gerät nicht) geben, der nur über eine manuell zu bedienende Taste oder einem Jumper zu beschreiben ist. So ist sichergestellt, dass keine unbemerkt eingedrungene Schadsoftware in der Lage sein kann den Speicherinhalt zu ändern. Dennoch gibt es die Möglichkeit Updates einzuspielen sofern eine berechtigte Person Vorort den dafür notwendigen Schalter betätigt. Dieser sollte übrigens optimalerweise mit wenigstens einem Schlüssel und/oder Codeschloss gesichert sein. Auch weitere Sicherheitsmaßnahmen wie biometrische Scanner können die Sicherheit an dieser Stelle erhöhen.

Ein Eeprom ist ein Festspeicher der über eine gesonderte Spannung gelöscht und neu beschrieben werden kann. Ein Eprom wird durch UV-Strahlung gelöscht und kann dann neu beschrieben werden. Natürlich ist auch ein sonstiger nichtflüchtiger Speicher denkbar wenn sein Beschreiben hardwareseitig blockiert ist bzw. nur durch manuellen Hardware-Eingriff möglich ist. Auch wäre denkbar dass die Chips ROM-Bausteine auf einem Sockel sind und der Software-Lieferant die Updates in Form von neuen ROM-Chips ausliefert. Ist die Echtheit dabei sichergestellt wäre das die sicherste Variante.

Es muss sichergestellt sein dass die Beschreibbarkeit nach Ausführen des Updates oder zumindest nach einer gewissen Zeit (z.B. 15 Minuten) automatisch wieder deaktiviert wird, falls das Zurücksetzen des Schreibmodus von Seiten des Anwenders vergessen wird.
Damit während dieses Schreibvorgangs Manipulationen (z.B. mit einem Rootkit/Bootkit) ausgeschlossen sind, könnte das Update eines solchen Speicherbausteins auch auf einem anderen System ausgeführt werden, welches standardmäßig nie am Netzwerk angebunden ist.
Selbstverständlich kann als Updateprogramm nur eines vom Hersteller der Chiffrierungssoftware verwendet werden welches die Integrität des erfolgten Updates mehrfach vergleicht.
Die Auslieferung von Updates, egal in welcher Form, muss strengen Sicherheitskriterien folgen, so dass Manipulationen auf diesem Weg ausgeschlossen werden können. Der Empfänger hat beim Hersteller stets nachzufragen und die Echtheit ist über Zertifikate nachzuweisen.
Mehrfache Prüfsummen stellen die Authentizität der Software sicher.

Idealerweise blendet sich dieser Speicher in einem bestimmten Speicherbereich des Arbeitsspeicher des Systems ein (ähnlich dem ROM/Flash für Bios). Hierzu muss ggf. eine kleine Änderung auf der Hauptplatine des Computers erfolgen. Es wäre auch ein Adapter vorstellbar, der zwischen Platine und RAM arbeitet und so Leseanforderungen auf einen bestimmten Speicherbereich auf das z.B. Eeprom umleitet.
Andernfalls wird die Software direkt aus dem z.B. Eeprom geladen und von verschiedenen

Programmen wie z.B. der Datenbankanwendung, einem gesonderten Vergleichsprogramm, dem Timer-Programm, speziellen System-Treibern usw. regelmäßig verglichen ob die Programme im Arbeitsspeicher noch mit denen im Festspeicher übereinstimmen, sodass Manipulationen des Programmcodes schnell entdeckt werden.

N. BESCHREIBUNG ANSPRUCH 9: (INTERRUPT-ZEIGER - HARDWARE-ERWEITERUNG)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
die o.g. Timer-Hardwareerweiterung oder eine andere Hardwareeinrichtung prüft, ob der zum Aufruf des Timer-Programms zuständige Interrupt-Zeiger manipuliert wurde, indem die Hardware kurz nach dem Auslösen des Timers eigenständig überprüft, ob der zuständige Prozessor auch tatsächlich in dem Speicherbereich arbeitet in dem das Timer-Programm gespeichert ist und andernfalls System-Alarm auslöst und/oder das System anhält.

Eine weitere Gefahr der Manipulation liegt in einer unberechtigten Veränderung des Interrupt-Zeigers, der auf das Timer-Programm zeigt (siehe Schritt c, Anspruch 1, bzw. Kapitel C.5). Dieser könnte auf ein Schadprogramm umgebogen werden, welches dann mit den durchaus in Erfahrung zu bringenden Verfahren und Parametern und der dann automatisch richtigen Systemzeit die Entschlüsselung der Futurekeys übernimmt und die Ausgangsschlüssel entwendet, oder Daten entschlüsselt und diese unerlaubt weiter gibt. Aus diesem Grund ist es entscheidend, dass dieser Interrupt-Zeiger davor geschützt wird bzw. Veränderungen zumindest sofort festgestellt werden.
Dies könnte durch die bereits mehrfach erwähnte Timer-Hardwareerweiterung geschehen (wenn diese eingesetzt wird). Wie im Anspruch 9 beschrieben prüft die Hardware ob nach dem Timer-Interrupt, also nach Auslösen des Timers, der Prozessor bzw. einer der Prozessoren, auch tatsächlich in dem Speicherbereich arbeitet, in dem das Timer-Programm gespeichert ist. Der Prozessor muss dazu immer wieder auf diesen Speicherbereich zugreifen. Dies wird aufgrund von heute üblichen Memory-Burst und größerem Prozessor-Cache sicherlich schubweise geschehen doch einige Operationen können nur erfolgen wenn der Prozessor auf die Adressen dieses Speicherbereichs zugreift. Dies kann eine Hardware erfassen indem sie die am Adressbus

angelegte Adresse „mit liest“ auch wenn sie nicht selektiert ist (siehe Fig. 7, Adress-Daten-Wandler). Dies muss dann ausgewertet werden. Ein Programm hinterlässt eine bestimmte „Signatur“, welche von der Hardware zu prüfen ist. Wurde beispielsweise - je nach System - 100 Prozessortaktzyklen nach dem Interrupt diese Programm-Signatur noch nicht erkannt, kann von einer Manipulation ausgegangen werden. Die Hardware reagiert dann mit verschiedensten Maßnahmen, wobei parallel zu einem Alarm wohl ein System-Halt das Sinnvollste sein dürfte damit ein möglicher Schaden sofort abgewendet oder zumindest begrenzt wird. Das System ist dann vom Netzwerk zu trennen und von Sicherheits-Fachleuten kontrolliert hoch zu fahren oder neu aufzusetzen.

Eine weitere Alternative wäre die Prüfung des sog. Prozessor-Stack.

Diese Prüfungen können entfallen wenn - was optimal wäre - der zuständige Interrupt(NMI)-Zeiger hardwareseitig fixiert ist sodass Manipulationen ohne manuellem Hardwareeingriff ausgeschlossen sind und dieser Interrupt(NMI)-Zeiger auf das Timer-Programm - am besten auf einem ROM untergebracht - zeigt.

Wird nicht mit der Timer-Hardwareerweiterung gearbeitet muss diese Prüfung anderweitig erfolgen. Siehe hierzu Kapitel R.2 “Prüfmechanismus für Software-Integrität“.

O. Weitere Dienste / Merkmale / Details zur Timer-Hardwareerweiterung

Um die bislang beschriebenen Funktionalitäten des Timers (z.B. Unlesbarkeit bis zum Auslösen etc.) zu gewährleisten muss dieser allerhöchstwahrscheinlich als Spezial-Timer auf einer gesonderten Hardware, der Timer-Hardwareerweiterung, dem System zur Verfügung gestellt werden. Siehe hierzu Anspruch 2, Kapitel D.

Um den vollen in diesem Verfahren dargestellten Funktionsumfang der Timer-Hardwareerweiterung sicher zu stellen, insbesondere auch den aus Anspruch 8 und Anspruch 9 sowie die nachfolgenden Dienste, wird die Timer-Hardwareerweiterung selbst wie ein kleiner Computer aufgebaut sein und eigenständige Programme ausführen können. Natürlich wird die Timer-Hardwareerweiterung keine Schnittstellen haben die ein Sicherheitsrisiko darstellen. Die Timer-Hardwareerweiterung wird einen Timer, einen Zufallszahlengenerator, einen eigenen Festspeicher (ROM), einen externen Festspeicher (Eeprom), und einen RAM-/Flash-Speicher

haben. Siehe Figur 7.
Sie sollte komplett abgeschirmt sein, was zusätzlich als Sicherungsgehäuse dient welches beim Öffnen eine Art Selbstzerstörung auslösen könnte.

Zur Erklärung der Figur 7:
Die Darstellung (Fig. 7) ist nicht vollständig sondern nur eine grobe schematische Übersicht, die einige wesentliche Bausteine und Verbindungen darstellen soll. Die Linien (Leitungen) von der Adress-Logik zu den jeweiligen Bausteinen beinhalten beispielsweise mindestens 2 Leitungen. Entweder Select und Read/Write oder gleich eine Read- und eine Write-Leitung.
Die Sel.-Logik sorgt dafür dass der Timer erst ansprechbar ist, nachdem er ausgelöst hat, womit das Alarm-Signal high ist. Dann aber nur ein mal. Die Alarm-Leitung setzt nämlich zusätzlich ein internes AND-Flip-Flop, welches jedoch durch die abfallende Flanke des Read-Select-Signals wieder resetet wird. Somit ist ein Lesen erst wieder nach dem nächsten Alarm möglich. Dies sind alles auch nur Ausführungsbeispiele. Gestaltungsmöglichkeiten gibt es viele um die immer wieder benannten und nachfolgenden Leistungsanforderungen der Timer-Hardwareerweiterung zu realisieren.

Zusätzlich zu den bereits aufgeführten Funktionen, insbesondere aus Anspruch 2, 8 und 9, soll die Timer-Hardwareerweiterung nachfolgende Funktionen / Dienste / Merkmale zur Verfügung stellen:

1. Hardwaremäßige Zufallszahlerzeugung

Die Problematik bei der Erzeugung von echten Zufallszahlen ist im Punkt K.1 (Zufallszahlen) bereits ausführlich dargestellt worden und es wurde auch bereits ausgeführt dass die Timer-Hardwareerweiterung aus diesem Grund einen hardwarebasierten nicht-deterministischen Zufallszahlengenerator höher Güte enthalten muss, um dem Verfahren maximale Sicherheit zu bieten.

a) nicht-deterministischen Zufallszahlengenerator

Der Zufallszahlengenerator der Timer-Hardwareerweiterung könnte auf dem Prinzip des thermischen Rauschens von Widerständen in Kombination mit Spannungsschwankungen bei Z-Dioden arbeiten. Dies in Kombination mit der kleinsten Zeiteinheit (Pikosekunden oder kleiner) eines Zählers gibt eine sichere Zufallszahl, insbesondere wenn der Taktgeber des Zählers keine so hohe Güte und damit höhere Schwankungen hat.
Aus dem Grund der Beeinflussbarkeit sind Generatoren auszuschließen, die ihre Werte aus dem

Rauschen bestimmter Funk-Frequenzen oder dem Stromnetz beziehen. Auch der Einfluss der Temperatur darf nicht zu vorhersehbaren oder wiederholbaren Ergebnissen führen. Selbstverständlich kann - zur Sicherheit - diese Hardware-Zufallszahl mit Software-Zufallswerten kombiniert werden, vorausgesetzt die Anforderungen aus Kapitel K.1 (Zufallszahlen), insbesondere die Manipulationssicherheit, werden erfüllt.

b) Parameter-Erzeugung

Zufallszahlen werden zum Großteil benötigt um kryptologische Schlüssel (Ausgangsschlüssel) zu erzeugen, wie z.B. den Zeitschlüssel. Das vorliegende Verfahren sorgt dafür dass diese Schlüssel vor unbefugten Zugriff geschützt sind. Dies muss jedoch auch sichergestellt sein, wenn es um Parameter geht.

Dies sind Zahlenteile einer Funktion oder Formel, welche z.B. die Berechnung der Veränderung von Masterkeys übernimmt. Könnte ein Angreifer diese Parameter manipulieren könnte das gesamte System ausgehebelt werden. Deshalb dürfen Parameter nur soweit eingesetzt werden wie jegliche Manipulation ausgeschlossen ist. Somit sollten wechselnde Parameter auf der Timer-Hardwareerweiterung gespeichert werden. Siehe hierzu die Ausführungen unter den nachfolgenden Kapitelnummern 7.b) ff.

c) Automatischer 192-Bit-Zeitcode und Zeit X

Eine Besonderheit besteht in der Erzeugung der Zeit X und des Zeitcodes durch die Timer-Hardwareerweiterung. Die Timer-Hardwareerweiterung könnte zusätzlich zur Zeit X, welche bei freien Alarmzeit-Register so wie auf der Timer-Hardwareerweiterung ein 128-Bit-Wert ist, eine (64-Bit-) Zufallszahl erzeugen welche außerhalb der Timer-Hardwareerweiterung nicht in Erfahrung zu bringen ist (siehe 7.b)) und die mit der Zeit X (128 Bit-Alarmzeit) multipliziert wird oder einfach als zusätzliche 64 Bit links angehängt, sodass ein 192-Bit-Wert entsteht; Der Zeitcode. Dieser wird dem Timer-Programm unmittelbar nach dem Setzen des Timers (damit dieser die Futurekeys damit verschlüsseln kann) und kurz nach dem Timer-Interrupt über die jeweils einmalig auszulesenden Zeit- und Alarmzeit-Register des Timers mitgeteilt. Dies verleiht dem Zeitcode echte 2^192 Möglichkeiten anstatt der 2^128 die er sonst hat und kann natürlich auch auf 256 Bit und bei entsprechender Anpassung noch weiter ausgebaut werden.

Ein Ablauf-Beispiel: Es wird eine 64-Bit- und eine 128-Bit-Zufallszahl erzeugt. Die 64-Bit-Zahl wird ins Zeit-Register geschrieben und die 128-Bit-Zahl ins Alarmzeit-Register. Die Timer-Hardwareerweiterung wartet nun bis die Register einmal ausgelesen wurden, was sie an einer And-Logik von Read- und Select-Signal des Timers erkennt. Dieses Read setzt die Select-Logik auf Inaktiv sodass weitere Reads nicht funktionieren. Nun wird das Zeit-Register auf den Wert

des Alarmzeit-Registers abzüglich der gewünschten Zeit X gesetzt und der Timer gestartet. Ab jetzt ist ein Auslesen sowieso gesperrt bis der Timer auslöst, also der Wert im Zeit-Register den Wert im Alarmzeit-Register erreicht und das Alarm-Signal „high“ wird. Durch letzteres wird der System-NMI-Interrupt ausgelöst, der das Timer-Programm aufruft, die Select-Logik frei geschaltet, sodass ein Lesen des Timers einmalig wieder möglich wird und das Zeit-Register auf den intern zwischengespeicherten 64-Bit-Wert gesetzt.
Das Timer-Programm liest diesen Wert und den Wert des Alarmzeit-Registers aus und hat einen echten 192-Bit-Zeitcode mit dem es die Futurekeys wieder entschlüsseln kann, ohne dass es einer parametrisierte Funktion zum Hochrechnen der Zeit X auf den Zeitcode bedarf.

2. Prüfmechanismus für Software-Integrität

Wenn systembedingt die direkte (per Einblendung) Ausführung eines auf z.B. Eeprom befindlichen Programmcodes (für u.a. dem Timer-Programm) nicht möglich ist oder gar ohne Hardwareerweiterung gearbeitet wird, prüft der Mikrocontroller auf der Timer-Hardwareerweiterung selbstständig und regelmäßig, in sehr kurzen Abständen, aber nicht vorhersehbar, die Integrität der sicherheitsrelevanten Programme, insbesondere des Timer-Programms.
Dies erfolgt zum Teil mit Prüfsummen aber auch per direkten Vergleich des Speicherinhalts des Arbeitspeichers im Adressbereich des Timer-Programms mit dem Timer-Programm auf dem Eeprom bzw. evtl. einer Kopie davon im ROM oder internen RAM der Timer-Hardwareerweiterung. Der Zugriff auf den PC-Speicher sollte per DMA (Direct-Mermory-Access) oder ähnlichen erfolgen.

In der Praxis dürfte eine Kombination aus gelegentlichem Komplett-Vergleich und deutlich häufigeren Prüfsummen-Check der beste Kompromiss aus Performance und Überwachung sein.

Somit wird auch für diese Ausführungs-Variante das Manipulationsrisiko minimiert und sicher gestellt dass Manipulationen erkannt werden, bevor sie nennenswerten Schaden anrichten können. Die Timer-Hardwareerweiterung reagiert wie üblich auf eine solche Gefährdung mit Alarm-Ton, System-Alarm und System-Halt.

3. Chiffrierungs- und Dechiffrierungsfunktionen

Die Timer-Hardwareerweiterung sollte wenigstens 2 symmetrische Ver- und Entschlüsselungs-Dienste und ggf. einen asymmetrischen zur Verfügung stellen wobei einer mit einem dauerhaften und unveränderbaren Schlüssel und der andere mit einem, per Initialisierung (durch das Timer-

Programm) von außen, mit auf der Hardwareerweiterung generierten Zufallszahlen erzeugten, Schlüssel operiert. Beide Schlüssel sollten mindestens 128-Bit-Schlüssel sein UND außerhalb der Hardwareeinrichtung unbekannt und unlesbar sein.

Diese Schlüssel sind eigentlich zur weiteren Verschlüsselung der Futurekeys und der Zeiger zu deren Verstecke vorgesehen, doch könnten Sie auch anderweitig genutzt werden. Für den Fall dass sie auch zur Verschlüsselung von Daten eingesetzt werden muss es dafür zusätzliche Schlüssel auf der Timer-Hardwareerweiterung geben, da diese dann auch außerhalb der Timer-Hardwareerweiterung gesichert werden müssten und dies für die ersten beiden Schlüssel nicht gestattet ist.

Die De-/ Chiffrierungsdienste (auch für die nachfolgenden Merkmale) können über den internen Mikrocontroller ggf. in Zusammenarbeit mit einer Floating Point Unit (FPU) wie in Figur 7 dargestellt und/oder über spezielle Krypto-Chips realisiert werden.

4. Automation

Je mehr des gesamten Datenschutz-Procedere auf der Timer-Hardwareerweiterung abläuft umso besser. Insofern sind folgende Varianten als zusätzliche Sicherheitssteigerungen zu sehen:

a) Timer programmiert sich selbst

Das Timer-Programm überträgt zu seinem Ende den Zeiger auf die Ausgangsschlüssel in der Matrix, sowie die Grenzwerte für eine neue Zeit X an die Timer-Hardwareerweiterung. Diese erzeugt mit dem eigenen Zufallszahlengenerator eine neue Zeit X und einen Zeitcode und setzt damit selbstständig ihren Timer. Der erzeugte Zeitcode kann dem Timer-Programm zum einmaligen Lesen über die Zeit-Register zur Verfügung gestellt werden.

b) Selbständige Verschlüsselung zu Futurekeys

Hierbei entnimmt die Timer-Hardwareerweiterung die Ausgangsschlüssel aus der Matrix, verschlüsselt sie mit dem Zeitcode und schreibt sie zurück oder stellt sie dem Timer-Programm an einer bestimmten Adresse des extern zugänglichen Speichers zur Verfügung oder speichert die so erzeugten Futurekeys im internen Schlüssel-Speicher auf welchem von außen nicht zugegriffen werden kann und der eventuell zudem Teil einer großen und langsamen Matrix ist.

c) Selbständige Entschlüsselung der Futurekeys

Sobald der Timer auslöst wird per Interrupt das Timer-Programm gestartet das die Matrix im Arbeitsspeicher neu vorbereitet. Ist dies fertig gestellt, gibt das Timer-Programm der Timer-

Hardwareerweiterung ein „Signal“ inklusive neuer Matrix-Zeiger für die Ausgangsschlüssel. Jetzt werden von der Timer-Hardwareerweiterung die internen Futurekeys mit dem Zeitcode (z.B. Alarm-Register-Stand des Timers) entschlüsselt und in die Matrix eingetragen.

d) Hardwareerweiterung übernimmt Chiffrierung

Eine weitere Steigerung der Automation besteht dann, wenn die Timer-Hardwareerweiterung die Ausgangsschlüssel überhaupt nicht mehr heraus gibt, sondern damit die Dechiffrierungs- und Chiffrierungsaufgaben selbst durchführt. Damit sicher gestellt ist dass die Anforderungen dazu auch wirklich vom Timer-Programm kommen, werden diese mit der letzten (aktuellen) Alarmzeit verschlüsselt oder bearbeitet hin und her übertragen. Es müssten dann jedoch auch die Parameter für die jeweilige arithmetische Bearbeitung des Masterkey's mit übertragen werden. Die Ausgangsschlüssel dürften dann jedoch nur im internen Arbeitsspeicher des Mikrocontrollers der Timer-Hardwareerweiterung abgelegt sein, so dass ein Diebstahl dieser technisch unmöglich ist. Eine entsprechende Prüfung der Karte müsste zudem feststellen wenn versucht wird, die Timer-Hardwareerweiterung unter Beibehaltung der Stromversorgung zu entfernen. Allerdings, - wen dies gelingen kann, könnte womöglich auch der gesamte Computer entwendet werden, was jedoch durch das optionale GPS-Modul der Timer-Hardwareerweiterung erkannt würde.
Ferner würde die Timer-Hardwareerweiterung natürlich jegliche Dechiffrierungsanforderungen außerhalb der Timer-Ruhe-Phase (Timer-Programm (exklusiv) aktiv) ablehnen und die zu de-/chiffrierenden Daten müssten mit dem letzten Alarm-Zeit-Register-Stand zumindest arithmetisch bearbeitet übermittelt und das Ergebnis damit auf andere Weise arithmetisch bearbeitet zurück übermittelt werden, damit ein Missbrauch des De-/Chiffrierungsdienst der Timer-Hardwareerweiterung verhindert wird.

Vorteile:
Der Hauptvorteil liegt natürlich darin dass die Ausgangsschlüssel und evtl. Futurekey so hardwaregestützt nicht mehr entwendet werden können. Diese werden nur bei Generierung von neuen Ausgangsschlüsseln einmalig verscrambelt heraus gegeben bzw. ausgedruckt. Zu Sicherungszwecken.
Bei entsprechender Gestaltung sollte auch ein Performance-Gewinn entstehen.

Nachteile:
Es funktioniert so nur mit maximal ausgestatteter Timer-Hardwareerweiterung inklusive Mikrocontroller.

Falls nun jemand den Eindruck hat, dies wäre dann fast wie ein TPM (Trusted Platform Module), - der irrt. Ein TPM hat eine ganz andere Zielstellung. Es ist nicht Teil eines Verfahrens und arbeitet insofern nicht mit einem Interrupt-Programm zusammen. Der Hauptschlüssel wird auch generell nicht heraus gegeben. Auch nicht zu Sicherungszwecken. Ein TPM setzt auf asymmetrische Verschlüsselung, bietet keinen Programm-Festspeicher und keine intelligente Speicherverwaltung und vor allem keine intelligente Zugriffskontrolle. D.h. ein TPM ist relativ leicht zu missbrauchen.

Gegenüber TPM und anderen Konzepten im klassischen Minni-Computer-Konzept ist wesentlich:

- Programme werden nicht vom RAM ausgeführt sondern ausschließlich vom ROM. Eine Manipulation bzw. ein Eindringen von Schadprogrammen ist ausgeschlossen.
- Der Zugriff von Außen ist hardwaremäßig sehr begrenzt.
- Die Kommunikation findet über eine Verschlüsselung / arithmetische Bearbeitung mit der Timer-Alarmzeit als ständig wechselnder, absolut geheimer Schlüssel statt.
- Adress-Zu-Daten-Wandler
- Masterkey-Verschlüsselung mit arithmetischer Veränderung
- Zeitschlüssel-Konzept integriert

Denkbar wäre auch, auf der Timer-Hardwareerweiterung eine SSD mit zu integrieren. Auf einer solchen SSD mit z.B. 1 TB Speicher könnte man über eine Milliarde Primzahlen mit jeweils ca. 1000 Dezimalstellen speichern. Diese könnten von der Timer-Hardwareerweiterung in ihren Leerlaufzeiten ständig erneuert werden. (Zum Thema G.3)

Die Automation könnte in der Tat soweit getrieben werden, dass es eines Timer-Programms nicht mehr bedarf, bzw. dessen Funktionen von der Timer-Hardwareerweiterung komplett selbst ausgeführt werden.

Es müssten dann eben dort diverse Algorithmen integriert werden um zu erkennen wenn statistisch ungewöhnliche Anforderungen eingehen, oder Dechiffrierungen angefordert werden zu Daten von Nutzern zu denen gar kein Login oder keine IP-Verbindung besteht, etc.

5. Adress-Kontrolle

Um Manipulationen vorzubeugen prüft die Timer-Hardwareerweiterung ob die Schreib- und auch Lesebefehle an sie für z.B. den Timer oder für Speicher der Timer-Hardwareerweiterung oder für Anforderungen von Chiffrierungs- oder Dechiffrierungs-Diensten auch tatsächlich aus dem Adressbereich des Timer-Programms erfolgen.

Dies wird erreicht indem der Mikrocontroller auf der Timer-Hardwareerweiterung ständig am Adressbus „mithört“ und die letzten beispielsweise 100 Adressen in einen roulierenden Speicher schreibt. Kommt nun ein Schreibbefehl an die Timer-Hardwareerweiterung kann diese Anhand der letzten Adressen feststellen, von welcher Speicher-Adresse dieser Schreibbefehl veranlasst wurde und dies mit dem Speicher-Bereich des Timer-Programm vergleichen. Stimmt dies nicht, löst die Hardware einen System-Alarm aus oder hält ggf. das System an um jegliche weitere Aktivität der offensichtlich vorhandenen Schadsoftware zu vermeiden.
Die heute übliche Burst-Technik macht zwar eine Zuordnung aktueller Schreib-/Lese-Zugriffe deutlich schwerer, aber auch dies ist, mit einem ggf. etwas längeren Adressdaten-Puffer möglich. In Figur 7 ist dieses Merkmal mit dem Baustein „Adress-Daten-Wandler“ dargestellt. Er stellt den Wert des Adressbus dem Datenbus zur Verfügung und/oder speichert beispielsweise die Werte der letzten 100 Takte.

Zusätzlich besteht immer die Möglichkeit, einen NMI/Interrupt auszulösen. Das Interrupt-Programm kann dann den Prozessor-Stack analysieren und so feststellen, welches Programm vor dem Interrupt abgearbeitet wurde.

Generell erfolgen Schreibbefehle an die Timer-Hardwareerweiterung so direkt wie möglich vom Timer-Programm und ohne dazwischen geschaltete Treiber oder ähnlichem. Ist dies systembedingt nicht möglich muss die Integrität der jeweiligen Treiber zusätzlich sichergestellt werden.

6. Diebstahl-Schutz

Ein ungewöhnliches aber nicht völlig undenkbares Angriffsszenario wäre der physische Diebstahl der Timer-Hardwareerweiterung, falls diese genutzt wird um Schlüssel zu speichern (wie im vorigen Kapitel Automation ausgeführt). Geschieht dies auf die übliche Weise, wären die Daten darauf geschützt, da sie mit Stromausfall verloren sind, vorausgesetzt es wird kein nichtflüchtiger Speicher verwendet. Darüber hinaus wird das Mikrocontroller-Programm der Timer-Hardwareerweiterung Techniken aufweisen, eine Entwendung festzustellen selbst wenn während dieses Vorgangs wesentliche Signale und die Stromversorgung sichergestellt bleiben. Diese Techniken müssen über kopierbare Hardware-ID’s und -Signaturen hinaus gehen und könnten u. a. ein GPS/Gallileo-Modul sowie empfindliche Gyro-Positions- und Beschleunigungssensoren beinhalten. Das Thema Abschirmung wird unter Q.7 aufgegriffen. Eine solche könnte jedenfalls auch genutzt werden, um einen Zugriff auf die Hardware der Timer-Hardwareerweiterung zu verhindern und sogar eine Art Selbstzerstörung beherbergen.

7. Flash-/RAM-Speicher

Die Timer-Hardwareerweiterung hat einen Lese-/Schreib-Speicher. (Fig. 7, RAM)

Je nach Anforderungen kann dies Flash-Speicher (oder vergleichbares) sein, der seine Daten auch bei Stromverlust behält, oder ein RAM (oder vergleichbares), der seine Daten bei Stromverlust verliert, was für bestimmte Szenarien ein wichtiges Sicherheitsmerkmal sein kann. In Figur 7 ist dieser Speicher mit RAM dargestellt was jedoch keine Festlegung auf diesen Typ Speicher bedeuten soll.

Zur Erklärung der Figur 7:
Die Adress-Logik 1 sorgt dafür, dass von dem RAM (oder sonstigem Lese-/Schreib-Speicher) nur ein kleiner Teil für einen Zugriff von außen zur Verfügung steht. Dieser Teil des Speichers dient auch zur Übertragung von Informationen und Parametern auf die Timer-Hardwareerweiterung und umgekehrt. In diesem Fall verbindet der PCI-Adapter den Bus der Timer-Hardwareerweiterung mit dem PC(I)-Bus. In allen anderen Fällen, insbesondere wenn der Mikrocontroller aktiv ist, wird über den PCI-Adapter der interne Bus vom PC(I)-Bus völlig getrennt oder zumindest der Adressbus. In jedem Fall sorgt die Adress-Logik (1) dafür, dass (nur) dann die entsprechenden Chips angesprochen werden, wenn die jeweiligen Adressbereiche angelegt sind und unterscheidet dabei ob dies von außen (PC-Bus) oder von innen (Mikrocontroller) erfolgt. Auf etwa 90% des RAM kann nur die CPU der Timer-Hardwareerweiterung zugreifen.

a) Sicherungen vor Defekten

Damit bei nicht redundanten Systemen die Ausgangsschlüssel vor Verlust durch einen System-/Hardware-Fehler/-Neustart geschützt sind, können diese (ohne Futurekey-Chiffrierung) auf der o.g. Hardwareerweiterung (z.B. auf einem Flash-Speicher, zusätzlich zum RAM der Timer-Hardwareerweiterung) gesichert werden, wobei diese Sicherung bzw. der Transfer zum Flash-Speicher ebenfalls nur verschlüsselt erfolgt mit Hilfe der angehaltenen Timer-Alarm-Zeit die nur dem Timer (und somit der Timer-Hardwareerweiterung) und dem Timer-Unterprogramm bekannt ist und wobei das Auslesen dieses speziellen Flash-Speichers grundsätzlich gesperrt ist und nur mit zusätzlichen Vorort am System auch hardwaremäßig zu erfolgenden Sicherheitsmaßnahmen, wie z.B. dem Betätigen einer Taste auf der Hardwareeinrichtung, möglich ist. Idealerweise wird ferner die Hardwareerweiterung vor Diebstahl geschützt, was u.a. mit Bindung an das jeweilige System (inkl. CPU-ID) erfolgen könnte.
Ein solcher Flash-Speicher könnte auch als Speicher der letzten z.B. 1000 Zeitschlüssel dienen

mit denen ein Backup erstellt wurde.

Ggf. sollte der Flash-Speicher mehrfach gespiegelt sein und zumindest einer davon eine galvanische Trennung zum restlichen System haben.

Ist jedoch das System nicht gegen alle denkbaren Katastrophen geschützt, macht diese Maßnahme keinen Sinn da in dem Fall ohnehin eine Sicherung der Ausgangsschlüssel wie im Kapitel T (Backup) dargestellt erfolgen muss. Die Sicherung auf der Timer-Hardwareerweiterung wäre dann ein unnötiges Risiko.

In diesem Fall ist es auch besser, den eben angesprochenen Speicher, nicht als Flash-Speicher, sondern als RAM auszuführen sodass die darauf befindlichen Daten bei Entwendung der Timer-Hardwareerweiterung verloren gehen, was insbesondere für das nachfolgende Kapitel wichtig wäre.

b) Parameter und Schlüssel mit Zugriffskontrolle

Der Flash-/RAM-Speicher könnte natürlich auch anderweitige geheime Daten zwischenspeichern. Insbesondere Parameter zu den anwendungsindividuellen Funktionen oder Zeiger. In dem Fall muss die Übermittlung verschlüsselt erfolgen, mit dem Timer-Alarm-Stand als Schlüssel (nur dem Timer und dem Timer-Programm bekannt) und die Timer-Hardwareerweiterung kontrollieren, ob die Schreib- und Lese-Befehle vom Timer-Programm kommen.

So könnte der Speicher der Timer-Hardwareerweiterung auch Futurekeys aufnehmen.

Den Zugriff darauf (so wie auch bei anderen Daten und Parametern) lässt die Timer-Hardwareerweiterung erst dann wieder zu, wenn das Timer-Programm läuft, was kurz nach dem Auslösen des Timers der Fall ist. Somit wäre sichergestellt dass kein anderes Programm Zugriff erhält, da das Timer-Programm exklusiv läuft. Zusätzlich kann auch hier der Lesezugriff jeweils auf einen einzigen (pro Timer-Periode) beschränkt werden.

Die Timer-Hardwareerweiterung stellt dies technisch sicher indem sie Daten denen jeglicher Zugriff von außen entzogen werden soll, ausschließlich im internen (Arbeits-)Speicher (RAM) hält und erst bei Zugriffserlaubnis auf die Speicherbereiche kopiert auf die von außen zugegriffen werden kann. Die Aufteilung dieser Bereiche ist fest in der Adress-Logik (1) der Timer-Hardwareerweiterung verbaut und kann nicht verändert werden. (Siehe Fig. 7) Auf den internen Speicher kann von außerhalb der Timer-Hardwareerweiterung nicht zugegriffen werden.

c) Besonders langsamer Speicher als Matrix

Zusätzlich könnte der Flash- oder RAM-Speicher oder ein Teil davon oder ein weiterer Speicher der von außen ansprechbar ist, hardwaremäßig so gebaut sein, dass er Leseanforderungen gewollt sehr langsam (z.B. < 100 Mbit/Sek.) ausführt. Somit könnte er genutzt werden um Schlüssel darin zu verstecken. Analog zu dem im Kapitel L. (Beschreibung ANSPRUCH 7: (HDD-Matrix – Versteck Futurekeys)) dargestellten Konzept, könnte ein Erbeuten eines kompletten Schlüssels innerhalb seiner Gültigkeitszeit unmöglich gemacht werden, wenn der Angreifer nicht weiß wo genau sich der aufgeteilte Schlüssel befindet. Dies kann eine sicherheitssteigernde Variante / Ergänzung zu der Matrix aus E.2. (Speicher-ZufallsMatrix) sein, aber auch als Alternaive für die im Kapitel L. (Beschreibung ANSPRUCH 7: (HDD-Matrix – Versteck Futurekeys)) genutzten Speicher herhalten.

Da das Timer-Programm nur den/die Schlüssel lesen muss und weiß wo sie sich befinden, ist deren Lese-Zugriff z.B. in 1,3 Mikrosekunden erfolgt, wohingegen der Scan / Diebstahl einer 1 GB großen Matrix 80 Sekunden dauern würde. Da z.B. die Ausgangsschlüssel nur für maximal ca. 3 Sek. dort verweilen, bevor sie wieder zu Futurekeys verschlüsselt werden, könnte ein Angreifer in dieser Zeit nur 1/25-stel der Matrix erbeuten. Der große Vorteil des Speichers auf der Timer-Hardwareerweiterung liegt aber auch darin, dass diese sicher stellen kann, dass Zugriffe nur erfolgen während das Timer-Programm (exklusiv) aktiv ist und zusätzlich kontrollieren könnte, aus und in welche System-Speicherbereiche die Schlüssel angefordert werden. Auch könnte die Timer-Hardwareerweiterung feststellen wenn jemand versucht, (wahllos oder systematisch) auf größere Mengen an Speicher zu zugreifen und entsprechende Maßnahmen ergreifen.

Der Sicherheitsgewinn gegenüber der Matrix im Arbeitsspeicher ist noch mal erheblich (wenngleich nicht notwendig aus heutiger Sicht). Man könnte (und müsste) die Lesegeschwindigkeit auf 500 Mbit/Sek. erhöhen und wenn die Timer-Hardwareerweiterung auch noch selbsttätig, während der Zeit in der das Timer-Programm nicht aktiv ist (Timer läuft), diesen Speicherbereich mit immer wieder neuen Zufallszahlen füllt, so könnte diese Variante unterm Strich keine großen Performance-Einbußen bedeuten. Aber natürlich ist klassischer Arbeitsspeicher schon deutlich schneller. Man muss sehen inwieweit der Prozessorcache das einigermaßen auszugleichen vermag.

Um diesen eventuellen Performance-Nachteil besser zu begegnen und trotzdem nicht an Sicherheit einzubüßen wurde das nachfolgende Verfahren entwickelt...

P. BESCHREIBUNG ANSPRUCH 10 (TRICKREICHER SPEICHER - HARDWAREERWEITERUNG)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
Speicher nur über einen Mikrocontroller / Speicher-Manager ansprechbar ist und dieser sicherstellt dass zufallsbestimmt ein Teil der Lese-Zugriffe auf diesen Speicher verzögert ausgeführt / beantwortet oder auch abgelehnt werden, außer es handelt sich um Lese-Zugriffe auf bestimmte Zellen dieses Speichers.

Der Mikrocontroller der Timer-Hardwareerweiterung fungiert hier wie ein Speicher-Manager, über den alle Speicherzugriffe zu einem speziellen Speicher laufen. Das bedeutet, ein externer Lesezugriff wird an dien Mikrocontroller geleitet, dieser wiederum schaltet die Hardwareerweiterung auf intern (PC(I)-Bus ausgekoppelt) und liest im Speicher, der hardwaremäßig fixiert nur so selektiert werden kann und stellt dann den gelesenen Wert extern zur Verfügung. Er stellt dabei sicher, dass die Lesegeschwindigkeit derart eingeschränkt bzw. verzögert wird, sodass es nicht möglich ist, den gesamten Speicher, oder einen bestimmten Bereich daraus, in angemessener bzw. für die darin gespeicherten Schlüssel oder Daten zur Gefahr werdenden Zeit, zu lesen. Er trägt aber auch dafür Sorge, dass Reads (Lesezugriffe / Lesebefehle) zu den Speicherzellen in denen die versteckten Schlüssel / Ausgangsschlüssel gespeichert sind, mit maximaler Geschwindigkeit laufen. Die Adressen dazu erhält der Mikrocontroller bzw. die Timer-Hardwareerweiterung über ein bestimmtes Verfahren. Im Falle der Ausgangsschlüssel kann er die Speicherorte einfach erkennen bzw. von Speicherbefehlen mit Zufallswerten unterscheiden: Es sind die ersten (bzw. letzen, falls zuvor die Matrix vom Timer-Programm mit Zufallszahlen gefüllt wird) Schreibbefehle nach einem Timer-Alarm-Event (nach dem Abruf der Timer-Alarn-Zeit). Dies sind die „bestimmten Zellen“ in der Definition von Anspruch 10. Da das Timer-Programm weiß, wo sich der/die Schlüssel befindet/befinden (Zeiger im Prozessorregister) würden Zugriffe auf diese Schlüssel immer mit maximaler Geschwindigkeit laufen wohingegen alle anderen sehr langsam wären. Als „bestimmten Zellen“ können jedoch auch eine bestimmte Anzahl Reads nach einem bestimmten Ereignis sein. So z.B. die ersten 512 Bit (z.B. die ersten 2 256-Bit-Schlüssel) die nach einem Timer-Alarm-Event gelesen werden. Was sehr effektiv funktioniert wenn die jeweiligen Schlüssel nur einmal pro Phase gelesen werden müssen, wie dies z.B. bei Futurekeys der Fall ist.

1. Trickreicher Speicher für RAM-Matrix

Dieser Geschwindigkeitsunterschied bei Reads würde jedoch verraten: Dies ist ein (Teil der) echten Schlüssel. Wenn jedoch die normale Lesegeschwindigkeit von 500 Mbit/sek. auf z.B. 1 Mbit/Sek. gesenkt wird, entsteht daraus zunächst keine Gefahr, denn der Angreifer käme gar nicht so weit um überhaupt mehr als einen Teil (z.B. 1/8) des Ausgangsschlüssel zu finden und diesen - statistisch gesehen - erst nach 62 Versuchen. Denn der Angreifer könnte in der z.B. 1 Sekunde in der die Ausgangsschlüssel dort präsent sind, maximal 1 Mbit und somit 1/500-stel einer 64 MB-Matrix lesen/testen. Doch vielfache Lesebefehle von Zellen der Speichermatrix, in denen die Ausgangsschlüssel nicht gespeichert sind, zeigen dem Mikrokontroller auch, dass hier ein Angriff im Gange ist. Die Timer-Hardwareerweiterung kann dann entsprechend reagieren.

Wohingegen bei der Matrix nach Anspruch 3 allein, die Sicherheit nur aus der Unauffindbarkeit in einer riesigen Menge an Zufallszahlen und somit Möglichkeiten resultiert, liegt die Sicherheit hierbei zusätzlich in der Unmöglichkeit relevante Größen des Speichers überhaupt erst mal zu lesen um den Schlüssel dann innerhalb einer großen Datenmenge finden zu wollen. Ein Füllen dieser Speichermatrix mit Zufallszahlen erübrigt sich jedoch nicht, da es sich um Ausgangsschlüssel handelt, die sich nicht ändern und ein Angreifer sich ansonsten Stück für Stück den richtigen Schlüssel zusammen klauen könnte. Dies könnte allerdings auch dadurch geschehen dass der Angreifer echte Schlüsselteile an der hohen Lesegeschwindigkeit erkennt. Um dies zu verhindern wird a) der Mikrocontroller, per Zufall ausgewählt, ca. 50% aller Reads ebenfalls mit maximaler Geschwindigkeit beantworten, sodass daraus keine Rückschlüsse auf echte Schlüsselteile gezogen werden können, aber dennoch ein Lesen der gesamten Speichermatrix völlig ausgeschlossen ist, - und b) die Größe der Matrix auf 128 MB erhöht um die 50% schnellen Reads wieder auszugleichen.
Die hier beispielhaft genannten 50% entsprechen dem „Teil“ in „zufallsbestimmt ein Teil der Lese-Zugriffe“ in der Definition zu Anspruch 10.

Der für die RAM-Matrix und damit Speicherung der Ausgangsschlüssel konkretisierte Verfahrens-Anspruch würde lauten:
Verfahren nach einem der vorhergehenden Ansprüche, wobei
ein spezieller Speicher für die Speicherung der Ausgangsschlüssel in einer RAM-Matrix nur über den Mikrocontroller der Timer-Hardwareerweiterung ansprechbar ist und dieser sicherstellt dass zufallsbestimmt im Durchschnitt 50% aller Lese-Zugriffe auf diese Speicher-Matrix so verzögert ausgeführt werden, dass die durchschnittliche Lesegeschwindigkeit 2 Mbit/Sekunde

beträgt, außer es handelt sich um Lesezugriffe auf die Zellen in denen die Ausgangsschlüssel(-Teile) gespeichert wurden.

Die Zufallszahlen der Speicher-Matrix müssen ständig erneuert werden, was ggf. die Timer-Hardwareerweiterung selbsttätig tun könnte; z.B. in den Pausen in denen das Timer-Programm nicht arbeitet und somit Reads auf diesen Speicher generell unterbunden werden.
Es muss hier - im Gegensatz zu der Matrix nach Anspruch 3 - Kapitel E.2 jedoch für eine einigermaßen gleichmäßige Verteilung der Schlüsselteile über die gesamte Speichermatrix, analog zu Kapitel L - Anspruch 7, gesorgt werden. Dies könnte jedoch der Mikrocontroller / Speicher-Manager auch selbstständig umsetzen sodass für das Timer-Programm dadurch kein zusätzlicher Aufwand entsteht.

Der Ablauf könnte folgendermaßen aussehen:
Das Timer-Programm schreibt einen Ausgangsschlüssel auf eine beliebige Speicher-Zelle der Matrix. Die Timer-Hardwareerweiterung, die bereits 0,3 Sekunden nach dem letzten Starten des Timers selbständig die komplette Matrix mit neuen Zufallszahlen gefüllt hat, fängt den Ausgangsschlüssel ab, teilt ihn in 8 Teile und bestimmt per Zufall wo in einer gedanklich geachtelten Speicher-Matrix die jeweiligen Teile gespeichert werden. Die jeweiligen Zeiger dahin merkt sich die Timer-Hardwareerweiterung in ihrem internen Speicher. Mit weiteren Speicherbefehlen verfährt die Timer-Hardwareerweiterung analog, wobei sie sich jeweils merkt, welche Speicheradresse des Speicherbefehls mit den tatsächlichen Speicherorten verbunden ist. Will das Timer-Programm wieder von der jeweiligen Stelle lesen, erkennt die Timer-Hardwareerweiterung das an der zwischengespeicherten Adresse und sucht anhand der damit verbundenen Real-Adressen die Teile des Schlüssels in der Speicher-Matrix zusammen und übergibt sie dem Timer-Programm.

Nun mag man vielleicht meinen, man braucht gar keine Matrix, denn die vom Timer-Programm verwendete Schreib-Adresse (im Prinzip der Zeiger, welcher nur im Prozessor-Register abgelegt werden darf), kann einfach wie ein Code gelten und wenn diese Adresse wieder angefragt wird, so gibt die Timer-Hardwareerweiterung den darunter gespeicherten aber in Wirklichkeit im internen Speicher abgelegten Ausgangsschlüssel wieder heraus. Oder es führt dann gleich damit De-/Chiffrierungen aus wie in Kapitel O.3, O.4, insbesondere O.4.d) ausgeführt.
Wird eine andere Adresse angefragt dann gibt die Timer-Hardwareerweiterung eine Zufallszahl zurück, die sie entweder (aus Zeitgründen) aus der Speicher-Matrix entnommen hat, oder frisch erzeugt hat. Letzteres würde einem Angreifer jedoch verraten, dass es ein Fake-Schlüssel war, wenn dieselbe Zelle ein zweites mal gelesen wird und dann ein neuer/anderer Zufallswert als

Ergebnis kommt. Gleichzeitig sollte jedoch ohnehin System-Alarm und Alarm-Ton ausgelöst werden, wenn versucht wird von einer Adresse zu lesen, die gar keinen Schlüssel gespeichert hat. Für einen System-Halt wäre das Gefahrenpotential jedoch zunächst nicht hoch genug. Nun, wenn die Adresse, die hier als Code gelten soll eine 64-Bit-Zahl ist, dann ist die Sicherheit in der Tat (fast) die Selbe, wie wenn ein Angreifer die Code-Funktion für den Matrix-Zeiger kennt und im Gegensatz zu der Speicher-Matrix im klassischen Arbeitsspeicher besteht keine Dump-Gefahr und die Timer-Hardwareerweiterung merkt wenn unbefugte Speicher-Zugriffe erfolgen.

Damit die Reads nicht komplexer sind als klassische Lesezugriffe auf den Arbeitsspeicher, wäre es ideal wenn die Timer-Hardwareerweiterung den virtuellen Speicher der Matrix in den jeweiligen Bereich des Arbeitsspeichers einblendet.

2. Trickreicher Speicher für HDD-Matrix

Aus Gründen der Performance ist - je nach System - eine Verwirklichung der Matrix nach E.2. bzw. Anspruch 3 auf der Timer-Hardwareerweiterung womöglich nicht sinnvoll, - als Speichermedium für Anspruch 7 ist ein entsprechend gestalteter Speicher auf der Timer-Hardwareerweiterung jedoch ideal. Es gibt dazu wenigstens 3 interessante Varianten:

- Die Timer-Hardwareerweiterung erhält 3 GB RAM, der ebenfalls nicht direkt ansprechbar ist, sondern nur über den Mikrocontroller der Timer-Hardwareerweiterung, welcher sich wie ein Speicher-Manager verhält. In dieser Eigenschaft sorgt er dafür dass der Lesedurchsatz bei z.B. 50 MByte/Sekunde liegt. Ähnlich zu d) könnte hier der Mikrocontroller die ersten 512 Bit Leseanforderung pro Timer-Phase mit maximaler Geschwindigkeit beantworten und erst dann langsam werden. Denn diese Schlüssel (Futurekeys) müssen nur einmal pro Timer-Phase gelesen werden.
- Die Timer-Hardwareerweiterung erhält z.B. eine 250 GB SSD. Diese ist allerdings nicht direkt ansprechbar, sondern nur über den Mikrocontroller der Timer-Hardwareerweiterung, welcher wiederum ein ähnliches Protokoll zur Verfügung stellt wie HDD's. Das bedeutet, - der Mikrocontroller tut so als wäre er eine HDD bzw. ein HDD-Controller und sorgt gleichzeitig dafür, dass die Lesegeschwindigkeiten unter der jeweiligen Begrenzung liegen. Auch hier könnten Techniken analog der o.g. eingesetzt werden um den Zugriff auf die echten Schlüssel zu beschleunigen.
- Der zu versteckende Futurekey wird dabei im internen Speicher der Timer-Hardwareerweiterung gehalten und ist somit von außen nicht greifbar, wobei der Mikrocontroller ihn dann frei gibt, wenn das Timer-Programm ihn wieder abruft. Speichern

und Abrufen erfolgt dabei mit einem 256-Bit-Code der als Schreibadresse genutzt wird. Dieser ist vergleichbar mit dem Zeiger der beim klassischen Verfahren nach Kapitel L benutzt wird. Diesen erzeugt - auf Anforderung - die Timer-Hardwareerweiterung per Zufallsgenerator und übermittelt ihn an das Timer-Programm welches ihn mit dem Zeitcode und/oder Zeitschlüssel verschlüsselt.

„Frei“ geben bedeutet, der Mikrocontroller überträgt den Schlüssel auf den extern verfügbaren Speicherbereich, wählt diesen an und dieser übergibt den Inhalt an den Datenbus, oder - je nach Möglichkeiten des Mikrocontrollers - übergibt dieser den Schlüssel direkt an Datenbus sodass er für den PCI-Bus wie das ganz normale Ergebnis eines Lese-Zugriffs wirkt.

Die Übertragung von und zur Timer-Hardwareerweiterung sollte jeweils mit dem Wert der letzten Alarm-Zeit verschlüsselt oder zumindest arithmetisch bearbeitet werden. Dies stellt sicher, dass beide Parteien sich gegenseitig authentifiziert wissen und die Übertragung nicht abgefangen werden kann.

Zusätzlich prüft die Timer-Hardwareerweiterung stets ob die jeweiligen Schreib- oder Lesezugriffe in den zeitlichen Rahmen passen, der aufgrund der unterschiedlichen Phasen veranschlagt werden kann. So muss unmittelbar (innerhalb von ca. 100 Prozessorzyklen) nach dem Timer-Alarm-Event-Interrupt der einmalige Lesezugriff für das Timer-Alarm-Register erfolgen (und ggf. für das Zeit-Register) und unmittelbar danach der Lesezugriff für den Futurekey. Der Schreibzugriff für den Futurekey erfolgt wiederum kurz nachdem der Timer neu gesetzt wurde. Jegliche Zugriffe außerhalb dieses Rhythmus oder gar mehrfache Leser- oder Schreib-Zugriffe zeigen eindeutig, dass hier ein Schadprogramm mitzumischen versucht. Parallel dazu kann die Timer-Hardwareerweiterung über das Mitlauschen am Adressbus immer kontrollieren ob das Timer-Programm arbeitet.

Zwar wird das vorliegende Verfahren hier nur auf Schlüssel angewandt, jedoch kann es letztlich für jegliche Daten verwendet werden.

Auch wenn hier, oder auch in einem Großteil der gesamten Patent-Beschreibung, vorwiegend von Flash-Speicher die Rede ist, soll der Speicher der Timer-Hardwareerweiterung nicht darauf festgelegt sein. Es kann stattdessen auch jeder andere Speichertypus Verwendung finden so lange er die jeweils geforderten Leistungen erbringt. So ist nur für einen geringen Teil die Nichtflüchtigkeit vorteilhaft. Ein Großteil des Speichers könnte und sollte als RAM umgesetzt werden oder in Form von Neuentwicklungen in diesem Bereich. Die Flüchtigkeit ist hier eher als Vorteil zu sehen.

Q. BESCHREIBUNG ANSPRUCH 11: (INDIVIDUALSCHLÜSSEL)

Verfahren nach einem der vorhergehenden Ansprüche, wobei
aus Passwörtern und/oder Anmeldenamen und/oder dessen Teilen und/oder Hash-Werten je ein zusätzlicher Einweg-Hashwert gebildet wird, der nicht dauerhaft gespeichert wird, sondern nur dazu dient, den zu dem jeweiligen Anmeldenamen / Passwort gehörenden Datensatz individuell zu verschlüsseln bzw. zu entschlüsseln.

Damit jeder Anwender sicher sein kann, dass selbst der Administrator des Datenbankservers, der sich illegal Zugriff auf die Ausgangsschlüssel verschaffen könnte, da diese aus Sicherungsgründen verscrambelt mitgeteilt wurden (siehe T „Backup"), nicht auf seine Daten zugreifen kann, wird nahegelegt, das nach Anspruch 4 (siehe Kapitel F - Beschreibung ANSPRUCH 4: (Datenschutz-System)) beschriebene Datenschutz-System wie nachfolgend zu ergänzen:

Beim Login wird zusätzlich zu den Hash-Werten welche zum Authentifizieren gebildet und benötigt werden ein weiterer kryptologischer Einweg-Hashwert gebildet. Für diesen wird das Passwort mit einem Schlüssel gehasht, wobei der dazugehörige Anmeldename oder ein Hash davon (ggf. abweichend von einem evtl. Standard-Anmeldedaten-Hash der anstatt des Klartext-Anmeldenamens an den Server übertragen wird) als Schlüssel verwendet wird und/oder ein anderes Verfahren angewendet wird als es für den Standard-Passwort-Hash zum Einsatz kommt. Dieser Zweit-Hash (Individualschlüssel) wird ebenfalls an den Server übertragen (nach den Richtlinien des Kapitel G) jedoch dort nicht dauerhaft (in der Datenbank) gespeichert. Er wird als temporärer kryptologischer Schlüssel (ähnlich wie ein Session-Key) benutzt um die Daten des zu diesem Anwender gehörenden Datensatzes individuell zusätzlich zu verschlüsseln bevor diese mit den üblichen Ausgangsschlüsseln z.B. nach Kapitel F (Beschreibung ANSPRUCH 4: (Datenschutz-System)) verschlüsselt werden.
Somit wird die richtige Kombination von Anmeldedaten und Passwort benötigt um die Daten wieder zu entschlüsseln. Selbst wenn jemand beispielsweise Masterkey und Zeitschlüssel hat, so könnte er zwar Zugang zu dem gespeicherten Passwort-Hash erlangen, da dieser jedoch ein Einweg-Hash ist, aus dem das ursprüngliche Passwort nicht wiederhergestellt werden kann, ist es

auch nicht möglich den Individualschlüssel in Erfahrung zu bringen. Dieser wird aber benötigt um die Daten gänzlich zu entschlüsseln.

R. Weitere Sicherungsmaßnahmen

1. Anwendungsindividuelle Funktionen

Funktionen, Formeln und Parameter, wie die Funktion für die Bearbeitung des Masterkey's nach Anspruch 4, sollen anwendungsindividuell sind.
Dies wurde bereits in den jeweiligen Kapiteln erwähnt. Es gilt insbesondere für die nachfolgenden Funktionen / Formeln

- Berechnung der Zeit X aus der Systemzeit
- Berechnung des Zeitcode aus Zeit X und ggf. weiteren Parametern
- Berechnung des jeweils veränderten Masterkey
- Berechnung des Verstecks für Masterkey-Futurekey
- Berechnung der Verstecke für Masterkey und Zeitschlüssel in der Matrix
- Berechnung wann neuer Zeitschlüssel zu generieren ist

Fest im Programmkern verbaute Funktionen sind z.B. durch Dekompilierung „relativ leicht" in Erfahrung zu bringen. Das stellt auch kein nennenswertes Problem dar. Vor allem auch deshalb, weil die meisten verwendeten Algorithmen, Funktionen, Formeln und Parameter, sowie die jeweils anzuwendenden Verschlüsselungs- und Hash-Verfahren und ggf. deren Schlüssel anwendungsindividuell sein sollten. Um sie in Erfahrung zu bringen muss also das jeweilige System befallen bzw. gekapert werden. Auch dies würde kein ernsthaftes Problem darstellen, ist aber ein zusätzliches Sicherheits-Feature.

Um die Herausforderungen für einen Angreifer weiter zu erhöhen, sollten so viele Parameter wie möglich in unbestimmten Abständen innerhalb bestimmter Grenzen variiert werden, was jedoch nur gestattet ist wenn diese völlig sicher vor Manipulationen z.B. durch Schadsoftware sind. Dies kann eigentlich nur realisiert werden wenn Parameter auf der Timer-Hardwareerweiterung gespeichert werden und diese Manipulationen unterbindet. Ein Beispiel dafür findet sich unter O.1. (Hardwaremäßige Zufallszahlerzeugung).
Es müssen sämtliche anwendungsindividuelle Formeln/Funktionen gegen Manipulation geschützt sein. Entweder werden sie bei Installation fest in den Programmcode übernommen

welcher per Festspeicher (z.B. Eeprom) fixiert wird oder zumindest unregelmäßig häufig per Prüfsummen gecheckt. So wie der gesamte sensible Programmcode. Wird die Software per z.B. ROM geliefert nimmt der Software-Hersteller die jeweiligen Individualisierungen der Funktionen und Parameter vor.

2. Prüfmechanismus für Software-Integrität

Als zusätzliche Sicherheitsmaßnahme, vor allem wenn systembedingt die direkte Ausführung eines auf z.B. Eeprom befindlichen Programmcodes nicht möglich ist, oder gar ohne Timer-Hardwareerweiterung gearbeitet wird, prüfen mehrere weitere Programme inklusive der Hauptanwendung regelmäßig (in sehr kurzen Abständen < 0,05 Millisekunden), zum Teil gegenseitig, ob es ggf. Manipulationen am Interrupt-Zeiger oder der jeweiligen (auch im Arbeitsspeicher aktiven / befindlichen) Software gab und löst notfalls System-Alarm aus und hält ggf. das System an. Dies könnte entweder durch kompletten oder stichpunktartigen Vergleich der im Arbeitsspeicher befindlichen (Chiffrierungs-)Programme mit den Programmdaten auf dem Eeprom oder durch Prüfziffer-Hashing erfolgen.
Diese Technik setzt darauf dass ein Schadprogramm nicht alle betroffenen Programme gleichzeitig manipulieren kann. Doch da dies andererseits auch nicht völlig ausgeschlossen ist, kann dieses Verfahren nur mit der Timer-Hardwareerweiterung als unknackbar gelten. Ggf. kann ein TPM abhelfen. Siehe 6.

3. Netzwerk

Hilfreich könnte auch sein, wenn das System während des Abarbeiten des Timer-Programms vom Netzwerk getrennt wird, bzw. sich vom Netzwerk trennt. Dies könnte softwareseitig oder für eine größere Sicherheit und Schnelligkeit hardwareseitig geschehen. Vorstellbar ist hier eine spezielle Funktion/Erweiterung der Netzwerkkarte bzw. deren Anbindung ans System um die Umschaltung innerhalb von Nanosekunden zu ermöglichen.

4. Dump

In jedem Fall muss bei dem ausführenden System die sog. Speicher- und Prozessor-Dump-Funktion deaktiviert sein und auch nicht aktivierbar sein. Auch für den Fall eines Prozessorfehlers/Absturz. Siehe hierzu die Ausführungen unter Kapitel E.2. „Speicher-ZufallsMatrix“.
Alternativ müssen die unter Kapitel P beschriebenen Verfahren angewendet werden.

5. Zeitangriff

Falls das Timer-Programm nicht exklusiv läuft oder es andere Gefahren gibt die eine Zeitnahme der Chiffrierungs- und Dechiffrierungs-Aktivitäten des Timer-Programm zulassen, muss explizit durch per Zufallszahl ausgewählte Zeitverschwendung sicher gestellt werden, dass aufgrund der messbaren Arbeitszeit keine Rückschlüsse auf Schlüssel oder Art und Umfang der Daten gezogen werden können.

6. TPM

Bei Betrieb des Verfahrens ohne Timer-Hardwareerweiterung kann die Integrität der verwendeten Programme und der Interrupt-Zeiger nicht durch die Timer-Hardwareerweiterung sichergestellt werden. Es sollte dann zumindest ein Trusted Platform Module (TPM) verwendet werden um die Integritätssicherheit des Systems (und der verwendeten Programme) zu erhöhen. Es muss dann ebenfalls der Interrupt-Zeiger für das Timer-Programm softwareseitig mehrfach kontrolliert werden.
Auch hier gilt es per Alarm und System-Halt, nennenswerte Schäden abzuwenden.

7. Abschirmung

Es ist auf verschiedene Arten möglich, dass Arbeiten eines Computer abzuhören. In verschiedenen Hoch- und Nieder-Frequenzspektren sind bestimmte Arbeitstypen und -Belastungen insbesondere der CPU hörbar zu machen. Oftmals tritt dies als Störung von Audio-Signalen auf. Zwar ist dem Erfinder kein solcher Angriff bekannt doch es ist nicht undenkbar dass daraus - ähnlich dem Zeitangriff - irgendwelche Rückschlüsse auf Chiffrierungsarbeit zu schließen sind. Das System muss daher entsprechend abgeschirmt und auch die Stromleitungen entkoppelt werden.

S. VERANSCHAULICHUNG (EINFACHE KONFIGURATION)

Gibt ein Anwender nun nach erfolgter Registrierung seine Anmeldedaten ein um Zugang zu der jeweiligen Website bzw. seinen erworbenen Diensten zu erlangen, so wird diese Anmeldedaten-

Anfrage zunächst in eine Warteschlange gestellt. Das in Anspruch 5 dargestellte Verfahren zum besonderen Schutz der Übermittlung von Anmeldedaten lassen wir hier außen vor.

Es gibt einen Web-Server mit Zugriff auf eine Datenbank (Datenbankserver) und das vorliegende mehrschichtige Schutzverfahren wird entweder in der Datenbankanwendung implementiert so dass die Anwender-Eingaben vor deren Verbindung/Weiterleitung zur Datenbank ausgefiltert werden und dann – sofern es sich um Anmeldedaten (oder anderweitige sensible Daten die zu sichern sind) handelt - durch dieses Schutzverfahren laufen bevor Sie der Datenbank übergeben werden. Gibt es nur einen Webserver könnte dieses Verfahren auch auf diesem zwischengeschaltet werden. Das Schutzverfahren wird im Übrigen auch sicherstellen, dass Anmeldedaten nie dechiffriert werden, da dafür auch keine Notwendigkeit besteht. Ferner wird es bei anderweitigen Dechiffrierungsanfragen (stichpunktartig) prüfen ob für die Anforderung ein gültiges Login besteht. Ggf. wird es hierfür eine interne (verschlüsselte) Liste führen um die Gefahr von SQL-Angriffen und Login-Täuschungen abzuwenden bzw. zu reduzieren.

Ein direkter Zugriff auf die Datenbank in Umgehung des vorliegenden Datenschutz-Verfahrens ist nicht vorgesehen, würde aber nur zur Erbeutung der mehrfach verschlüsselten Daten führen und wäre daher nutzlos. Die Gefahr bei SQL-Angriffen ist die, dass der SQL-Server eine etwaige Verschlüsselung vor Ausreichung der Daten selbsttätig entschlüsselt, was hier jedoch so nicht passieren kann. Das Datenschutz-Verfahren kann nicht nur prüfen ob gültige Login's bestehen sondern auch sicherstellen, dass nicht mehr als üblich innerhalb bestimmter Zeiträume ausgereicht wird und sicherstellen dass die IP's der jeweiligen Clients unterschiedlich sind.
Es gibt mindestens zwei Ausgestaltungsmöglichkeiten:

- Der SQL-Server wird ganz normal angesprochen und sendet dann über die Pufferdatei die Verschlüsselungs- bzw. Entschlüsselungsanfragen an das Datenschutzverfahren. Nachteil dabei: Es könnten rein theoretisch unverschlüsselte Daten abgespeichert werden wenn der SQL-Server „vergisst" diese vorher verschlüsseln zu lassen. Ferner müsste der Datenbankserver angepasst werden.
- Die Datenbank-Engine (Schnittstelle des Webservers zur Datenbank) wird angepasst und fängt SQL-Befehle zum Lesen und Schreiben ab, leitet diese ans Datenschutzverfahren und dieses gibt dann nach Entschlüsselung oder Verschlüsselung die Daten an den Datenbank-Server weiter.

1. Passwörter kommen i.d.R. bereits 2-fach gehasht (vom Frontend) an oder werden per teilweise verkürzten Einweg-Hashing sofort zu unwiederherstellbaren Hash-Werten verschlüsselt.

2. Anmeldename, Passwort und Daten werden an das Schutzverfahren übergeben. Dies erfolgt indem die Daten als neuer Eintrag einer speziellen (Stack-)Datei hinzugefügt werden. Er beinhaltet den Chiffrierungsgrund, die Anmeldedaten und eine Zuordnungsnummer der Anfrage. Ggf. dazugehörige weitere Daten werden ebenfalls gepuffert. Selbstverständlich wären auch andere Wege des programmübergreifenden Datenaustauschs möglich, sofern sie sicher sind.

3. Zur Zeit X ruft der wiederholt gesetzte Timer (per Interrupt) das entsprechende Timer-Programm auf.

4. Dieses blockiert sofort weitere Interrupts und speichert die Systemzeit sowie die Timerwerte (Nachlaufzeit zum Ausgleich der Zeitverzögerung seit Timer-Ablauf, bzw. Zeitcode-Basis) und stoppt dann ggf. die Netzwerkverbindung zum Web(-Server) und weitere Threads.

5. Nun wird die Zufallsmatrix zur gesicherten Ablage der Schlüssel vorbereitet und aus der gesicherten Timerzeit / Systemzeit (ggf. abzgl. des Timerwertes) der Zeitcode und daraus der gültige Zeitschlüssel und der Masterkey berechnet und ggf. die Fehlerkorrektur angewendet. Zum Auffinden des Masterkey-Futurekey's wird der Zeiger darauf zuvor mit dem dechiffrierten Zeitschlüssel entschlüsselt. Der Zeitschlüssel-Futurekey und der Masterkey-Futurekey wird (im Versteck) sicher gelöscht. Ebenfalls der Zeitcode.

6. Dann werden die Daten aus der (Stack-)Datei entnommen, dechiffriert und sogleich dort sicher gelöscht.
   Der Anmeldename wird mit dem Masterkey und Zeitschlüssel verschlüsselt und das Ergebnis als Suchanfrage an die Datenbank gesendet. Bei positiven Ergebnis wird der Masterkey mit Klardaten, der Datenfeldnummer und des Unique-Key's des Datensatzes gemäß der anwendungsindividuellen Formel arithmetisch bearbeitet und der Passwort-Hash damit verschlüsselt. Anschließend erfolgt die Verschlüsselung mit dem Zeitschlüssel und das Passwort wird mit dem der Datenbank verglichen (bzw. bei Registrierungen dort gespeichert). Ggf. können weitere sensible Daten (aus dem Puffer) mit einem jeweils neu arithmetisch zu bearbeiteten Masterkey und dem Zeitschlüssel chiffriert gespeichert werden.

Das Login wird in Verbindung mit der Zuordnungszahl des Datenbanksatzes und der IP codiert und dieses mindestens mit dem Masterkey verschlüsselt in eine Login-Datei abgelegt.

7. Der Chiffrierungsgrund der (Stack-)Datei entscheidet die exakte Vorgehensweise. Er startet u.U. auch weitere Aktivitäten wie z. B. die Erstellung einer Datensicherung.

8. Die o.g. (Stack-)Datei erhält u.U. unter der jeweiligen Zuordnungsnummer eine Quittierung/Rückmeldung.

9. Eventuelle Zwischenspeicher (Variablen) für die aus der Datei entnommenen Anmeldedaten werden sicher gelöscht.

10. Es wird geprüft ob eine Zeitschlüssel-Umchiffrierung der Datenbank noch nicht abgeschlossen ist und dieses dann z.B. 1 Sek. weiter ausgeführt. Ist dies abgeschlossen wird berechnet ob der Zeitschlüssel neu generiert werden soll und das ggf. gemacht. Der alte Zeitschlüssel wird solange als gültig beibehalten bis die Umchiffrierung abgeschlossen ist.

11. Das neue Versteck für den Masterkey-Futurekey wird (zufällig) ausgewählt. Der Zeiger darauf wird mit gültigem Zeitschlüssel chiffriert, wobei das Original noch nicht gelöscht wird.

12. Es wird die neue Zeit X festgelegt und daraus ein 192-Bit-Zeitcode errechnet. Dies geschieht mit extrem komplexen Verfahren aus Verschlüsselung und Potenzierung mit komplexen vielstelligen Zahlen. Daraus wird dann je ein neuer Futurekey berechnet und der Timer entsprechend gesetzt.

13. Masterkey und Zeitschlüssel werden sicher gelöscht.

14. Ggf. wird die Netzwerkanbindung / Internetanbindung wieder hergestellt.

15. Der Masterkey-Futurekey wird versteckt und das Original sowie der unchiffrierte Zeiger darauf sicher gelöscht.

16. Blockierte Threads und Interrupts werden wieder aktiviert und ggf. der Web-(Server-)Anwendung eine Botschaft gesendet, dass die Anmeldedaten bearbeitet bzw. ausgewertet wurden, sodass evtl. weitere gepufferte Anfragen weiter geleitet werden.

Zwar wird hier in erster Linie über Anmeldedaten gesprochen, aber selbstverständlich lassen sich mit diesem Verfahren auch alle anderweitigen Daten schützen. Zu empfehlen ist jedoch dass größere Mengen sensibler Daten stets mit einem gesonderten Masterkey/Zeitschlüssel verschlüsselt werden und wenn sie (das jeweilige Datenfeld bzw. die jeweilige Datenspalte) nicht für eine datenbankweite Suche zur Verfügung stehen müssen, sollte unbedingt die arithmetische Bearbeitung des Masterkey's (vor der weiteren Verschlüsselung per Zeitschlüssel) erfolgen. Denn ist eine Datenbank mit einem einzigen Schlüssel verschlüsselt, gilt: Je größer die Datenmenge desto leichter ist der Schlüssel zu knacken.

Selbstverständlich haben hochwertige Datenbanksysteme eine eigene Verschlüsselung. Diese ist bei Anwendung des vorliegenden Verfahrens eigentlich überflüssig, schadet aber auch nicht. Insbesondere muss sich somit die Anwendung des vorliegenden Verfahrens nicht um die Verschlüsselung der einfacheren und nicht ganz so sensiblen Daten kümmern. So würde es beispielsweise bei einem Bankserver ausreichen, die Anmeldedaten und z.B. Kontonummer sowie Strasse mit dem vorliegenden Verfahren zu schützen, wobei für Kontonummer und Strasse je ein gesonderter Ausgangsschlüssel zum Einsatz kommen könnte. Alle restlichen Daten wären mit der Datenbankverschlüsselung ausreichend gesichert da mit ihnen im Fall der Erbeutung kein dramatischer Schaden entstünde.

T. BACKUP (UND SCRAMBLE-CODES)

Es ist davon auszugehen dass von der Datenbank z.B. täglich eine Sicherung (BACKUP) erstellt wird/werden soll. Dies erfolgt zumeist auf einem gesonderten Sicherungslaufwerk. In diesem Fall wird entweder die Datenbank mit Hilfe des Zeitschlüssels, da dieser nach kurzer Zeit verworfen sein wird, dechiffriert und dann gesichert (also „nur" mit dem arithmetisch veränderten Masterkey verschlüsselt) oder es wird der für diese Sicherung gültige Zeitschlüssel ausgedruckt bzw. dem Anwender mitgeteilt sodass das Backup notfalls wiederhergestellt werden kann. Da der gültige Zeitschlüssel außerhalb des Timer-Programms unbekannt ist und erst zum nächsten Timer-Ablauf aus dem Futurekey und der Systemzeit berechnet werden kann, muss diese Sicherungsfunktion vom Timer-Unterprogramm ausgeführt werden.

Damit der ausgedruckte oder mitgeteilte Backup-Zeitschlüssel nicht außerhalb des Systems, also in der materiellen Welt (womöglich zusammen mit dem dazugehörigen Backup z.B. aus einem

Safe) entwendet werden kann, wäre es sinnvoll, dass das Timer-Programm die zu den (beispielsweise 100 letzten) Backups gehörigen Backup-Zeitschlüssel in einer Liste speichert die so wie die Anmeldedaten auch mit dem kompletten Schutzverfahren chiffriert sind und somit nicht erbeutet werden können. Falls ein Backup wiederhergestellt werden muss wird dies dem Timer-Programm (so wie auch der Backup-Vorgang) über eine spezielle Anfrage mitgeteilt und nach dessen Anweisung das Backup eingespielt. Dem Timer-Programm wird das Datum des Backups mitgeteilt sodass es den dazugehörigen Backup-Zeitschlüssel aus der o.g. Liste extrahieren kann.

Möglichen Hardware-Defekten kann mit doppelten Systemen / Redundanz abgeholfen werden. Nicht jedoch anderen (weltlichen) Katastrophen. Aus diesem Grunde sollten Sicherungen auch nie am selben Ort wie die Quell-Datenbank aufbewahrt werden oder nur in einem extrem gesicherten Datenträger-Safe. Für solche Fälle müssen die Ausgangsschlüssel - soweit sie für die Entschlüsselung der Datenbanksicherungen notwendig sind - ebenfalls an einem sicheren Ort hinterlegt sein. Das Timer-Programm bietet die Möglichkeit diese auf verschiedene Art auszudrucken. Eine davon ist eine Scramble-Methode zu der z.B. zwei Code-Blätter ausgedruckt werden, die an verschiedenen Orten aufzubewahren sind. Nur wenn man beide übereinander gelegt gegen eine starke Lichtquelle hält kann man den enthaltenen Schlüssel lesen. Zu empfehlen ist dies jedoch mit 3 oder 4 Lagen / Blättern umzusetzen, was die Sicherheit deutlich erhöht. Zur Dechiffrierung müssen diese dann auf Folie kopiert und übereinander gelegt oder eingescannt und digital übereinander gelegt werden. Hierbei gibt es Methoden mit einfarbigen und mehrfarbigen Ausdrucken, wobei letztere mehr Möglichkeiten des Verschleierns bieten. Auf diesen Notfall-Code-Blättern müssen ebenfalls alle anwendungsindividuellen Parameter und Funktionen untergebracht sein, soweit diese z.B. bei einem Brand verloren gehen würden. Natürlich wäre für die Sicherung dieser Werte auch eine vollständige oder teilweise Online-Sicherung denkbar. Doch dabei sollte als Übertragungssicherheit zumindest dieselben Verfahren angewandt werden, wie für Passwörter unter G.3 dargestellt. Für die Sicherheit der übermittelten Werte sollte dann wiederum das vorliegende Verfahren sorgen. Da das Risiko aufgrund des etwaigen (rein theoretischen) Schadens deutlich höher liegt sollten die Schlüsselgrößen hier jedoch deutlich größer (mindestens doppelt so groß) sein. Die zusätzliche Rechenzeit kann dabei akzeptiert werden da niemand darauf warten muss.

Bezüglich des Zeitschlüssels werden - wie bereits angeschnitten - entweder zu jedem Backup die Zeitschlüssel-Verschlüsselung entfernt oder die Zeitschlüssel per neuen Scramble-Code ausgedruckt. Dieses enthält dann natürlich auch das Datum des Backups.

Gibt es einen System-Ausfall oder -Fehler mit anschließenden Neustart (Reset) sind die im System gespeicherten Schlüssel verloren. Insbesondere der Zeitschlüssel und der Zeitcode zur Dechiffrierung der Futurekeys und auch die Zeiger auf deren Verstecke. Es müssen dann nach dem Neustart die Ausgangsschlüssel neu eingegeben werden. Da es nicht praktikabel ist, für den Zeitschlüssel bei untertäglicher Erneuerung jedes Mal einen neuen Scramble-Ausdruck zu machen und aufgeteilt in Sicherheit zu bringen, müsste in diesem Fall das letzte Backup wieder eingespielt werden.
Dies gilt nicht unbedingt, wenn die Timer-Hardwareerweiterung selbst die De-/ Chiffrierungsaufgaben übernimmt und die Schlüssel speichert, wobei dies nur empfehlenswert ist, wenn die Timer-Hardwareerweiterung gegen physische Entwendung gesichert ist.
In diesem Fall kann auf der Timer-Hardwareerweiterung eine Drucker-Schnittstelle untergebracht werden sodass darüber die Scramble-Codes direkt ausgedruckt werden können.

Auf eine Sicherung etwaiger zusätzlicher Hardware-Schlüssel des Chiffrierungssystems der Timer-Hardwareerweiterung außerhalb dieser (wegen etwaiger Hardware-Defekte) kann verzichtet werden wenn diese Verschlüsselungsstufen bei einem Backup ausgelassen bzw. entfernt werden, was wiederum nur zu empfehlen ist wenn die Backups besonders sicher aufbewahrt werden.

U. ALTERNATIVEN VERSAGEN

1. Generell asymmetrische Verschlüsselung

Man könnte vielleicht meinen, dass mit Hilfe z.B. einer RSA-Verschlüsselung (asymmetrische Schlüssel) und Aufbewahrung des privaten/geheimen Schlüssels an einem anderen sicheren Ort eine ähnliche Sicherheit erreicht werden könnte, da der evtl. mit erbeutete (öffentliche) Schlüssel nicht ausreicht um die Daten zu entschlüsseln (was normalerweise auch gar nicht nötig ist sofern es sich nur um Anmeldedaten handelt). Doch aufgrund der rasanten Entwicklung im Bereich der Rechenleistung ist heute schon klar, dass selbst Schlüssel mit 2048-Bit Länge in naher Zukunft berechnet werden können (Faktorisierung) sofern der öffentliche Schlüssel bekannt ist. Insofern wäre hiermit eine echte oder gar langfristige Sicherheit definitiv nicht gegeben. (Siehe Schätzungen und Prognosen von BSI, NIST, bzw. Dirk Fox (Secorvo GmbH)) Vor allem wäre

jedoch diese Handhabung nicht zweckmäßig wenn auch (sensible) Daten wieder dechiffriert werden müssen. Die Erbeutung des privaten Schlüssels wäre ferner genauso wie bei jedem symmetrischen Schlüssel nie ganz auszuschließen.

2. Einweg-Hashing

Es bestünde ferner die Möglichkeit allein mit einer starken Einweg-Verschlüsselung die reale Entschlüsselung erbeuteter Daten selbst mit dem richtigen Schlüssel auszuschließen, wie dies heute selbst bei großen Unternehmen (z.B. Yahoo) praktiziert wird. Allerdings ist es in diesem Fall möglich Äquivalentwerte zu finden, die zu dem gleichen Ergebnis (Hashwert) kommen und somit Zugriff gewähren obwohl der ursprüngliche reale Ausgangswert nicht wiederhergestellt werden konnte. Zusätzlich sind 99% aller Passwörter mit Brute-Force- oder Wörterbuchangriffen relativ schnell zu finden.
Ferner wäre auch hier das Problem dass es natürlich auch Daten gibt die nicht mit einem Einweg-Hashing verschlüsselt werden können da es notwendig ist sie wieder völlig herstellen zu können.

3. Trusted Platform Module

Zwar wäre die generelle Verschlüsselung per TPM (Trusted Platform Module) ebenfalls möglich und somit wäre auch sichergestellt dass der Schlüssel nicht ohne weiteres erbeutet werden kann, sofern hierfür der Endorsement-Key verwendet würde, da dieser (der private Teil) ausschließlich dem TPM bekannt ist, doch wäre aufgrund dessen dass dieser immer gleich ist, ein Angriff z.B. per Kryptoanalyse durchaus denkbar und je nach Rechenleistung womöglich sogar in vertretbarer Zeit. Zusätzlich bestehen noch das Problem bei einem Hardware-Defekt und die Gefahr einer Analyse des Schlüssels bei direkter Nutzung der TPM-Verschlüsselung. So könnte ein Angreifer immer wieder verschiedenste Werte vom TPM ver- und entschlüsseln lassen und daraus analysieren, welche Veränderungen des Eingangswertes zu welchen Veränderungen des Ausgangswertes führen.
Vor allem jedoch gelten asymmetrische Verschlüsselungen heute schon als brechbar, wenn ausreichend Zeit vorhanden ist. Insgesamt würde der Funktionsumfang eines TPM für den Großteil der hier vorliegenden Sicherheitsmechanismen nicht ausreichen.

Auch kann sich ein TPM nicht gegen Missbrauch wehren, so könnte ein Schadprogramm das TPM zum Entschlüsseln der Daten missbrauchen.

Ein TPM bietet auch nicht die hier dargestellten Methoden zur Sicherung gegen Hardware-Fehler.

Das hier dargestellte Verfahren ist ein Gesamtkonzept welches vor allem in der Verbindung von Timer-Hardwareerweiterung und Timer-Programm unschlagbar ist. Die stets wechselnden Schlüssel und die arithmetische Veränderung dieser sind ebenfalls entscheidende Unterschiede und Vorteile.

V. ERGEBNIS

Mit diesem Verfahren geschützte Daten/Passwörter können bei heutigem Wissensstand und absehbaren Technikstand nicht erbeutet werden. Selbst bei einem kompletten Datenklau und einer Muster-Kryptoanalyse und/oder einem Brute-Force-Angriff eines Super-Computers auf die erbeuteten Daten kann ein Erfolg in den nächsten 150 Jahren ausgeschlossen werden und bei entsprechender Schlüsselstärke auch deutlich länger. Ein Online-Angriff auf die Datenbank direkt kann aufgrund der mehrfachen inhomogenen Verschlüsselung ausgeschlossen werden.

Es bleibt am Ende das Hauptrisiko, dass die Ausgangsschlüssel erbeutet werden, was jedoch durch das vorliegende Verfahren ausgeschlossen werden kann da sämtliche relevanten Schlüssel niemals wirklich vorliegen. Hauptsächlich sind nur die Futurekeys vorhanden, welche - selbst wenn sie erbeutet würden - nur mit Brute-Force bei ebenfalls entwendeter Datenbank angegriffen werden könnten. Doch aufgrund der eingebauten zeitlichen Brisanz ist die Erbeutung beider Schlüssel ausgeschlossen. Wie man es auch dreht und wendet. Am Ende könnte nur das Durchspielen von mindestens 2^256 Möglichkeiten (bei der Minimums-Anforderung der Schlüssellängen). Will man auch nach 150 Jahren noch auf der sicheren Seite sein, nimmt man 2 256-Bit-Schlüssel und die Daten sind 400 Jahre sicher. Bei 2 384-Bit-Schlüssel z.B. sind es schon 650 Jahre bevor ein Top-Ten-Super-Computer in der Lage sein dürfte, alle Möglichkeiten in einigen Jahren zu knacken. Aus heutiger Sicht übersteigt die nötige Rechenzeit noch lange Alter und Lebensdauer des Universums.
Wann die Futurekeys Gültigkeit erlangen ist nicht herausfindbar und während der kurzen Ablaufdauer des Timer-Unterprogramms liegen Masterkey und Zeitschlüssel nur so versteckt vor dass es selbst mit Super-Computern in 1000 Jahren unmöglich sein dürfte die tatsächlichen

Schlüssel heraus zu finden. Der codierte Zeiger darauf liegt während dessen nur in Prozessorregistern, auf die selbst mit einem Hardwareeingriff nicht zugegriffen werden kann. Masterkey und Zeitschlüssel sind quasi wie Geister der Zukunft, - nicht da und wenn doch dann nicht zu sehen. (Daher die Bezeichnung Ghosten)

Selbst das latente Risiko einer Erbeutung eines kompletten Speicher- und Prozessor-Dump (inklusive aller Prozessor-Register) ist bei Anwendung der Verfahrensschritte aus O.7.c) oder des entsprechenden Einsatzes der Timer-Hardwareerweiterung ausgeschlossen.

Ergänzend können noch weitere Verschlüsselungsstufen mit wechselndem Schlüssel über die Timer-Hardwareerweiterung eingefügt werden, aber die in Kapitel O.3 (Chiffrierungs- und Dechiffrierungsfunktionen) dargestellten Methoden sind völlig ausreichend, sodass letztlich auch ein Speicher- und Prozessor-Dump nicht zu einem erfolgreichen Angriff führen kann, da (zum Teil) die De-/ Chiffrierung in der Hardwareeinrichtung stattfindet und diese Schlüssel dort unlesbar untergebracht sind. Sie werden auf der Hardwareeinrichtung erzeugt und verlassen diese nicht. Da die Schlüssel wechseln und zusätzlich die Timer-Hardwareerweiterung zu einem unveränderlichen Schlüssel noch einen variablen innehält der ähnlich wie der Zeitschlüssel in gewissen Zeitabständen neu erzeugt wird, kann eine Krypto-Analyse ausgeschlossen werden.

Somit sind die Daten selbst dann absolut sicher, wenn der jeweilige Server massiv infiziert ist und Schadprogramme unbeschränkten Zugriff erlangen.

Manchen mag der Aufwand dieses Verfahrens vielleicht als übertrieben erscheinen.
Nun, - leider lernen die Menschen (fast) nur aus Fehlern bzw. aus Schmerz. In der Vergangenheit wurden die meisten Innovationen im Sicherheitsbereich immer erst erfunden, wenn ein erfolgreicher Angriff eine Schwachstelle offenkundig gemacht hat und somit eine Weiterentwicklung notwendig wurde.
Das ist hier nicht der Fall. In diesem Verfahren sind von vorneherein alle möglichen Angriffszenarien berücksichtigt, auch wenn manche (wahrscheinlich) noch nie gelungen sind. Es sind so viele Sicherungsschichten vorhanden, dass ein erfolgreicher Angriff auf lange lange Zeit ausgeschlossen werden kann. Wenn die entsprechenden Verfahren und Schlüsselgrößen stets angepasst und anspruchsvoll gehalten werden, vielleicht sogar ewig.

Da Anmeldedaten generell nie mehr entschlüsselt werden, könnte ein Angreifer maximal 4-fach verschlüsselte Daten erlangen zu dessen Entschlüsselung die Lebensdauer des Weltalls nicht ausreicht.

W. SIEGEL

Erfüllt ein Server alle wesentlichen Sicherheitsbausteine so erhält er ein SIEGEL, das er u.a. auf seinem Portal veröffentlichen kann und welches (ähnlich dem „Trusted-Shop-Garantie-Siegel") dem Anwender / Nutzer die Gewissheit gibt, dass seine (Anmelde-) Daten hier sicher sind. Hierbei ist eine mindestens zweistufige Qualitäts-Klassifizierung sinnvoll um zu unterscheiden ob das Maximum der Sicherheitsmechanismen (inklusive Hardwareerweiterung oder TPM und Handhabung der jeweils sichersten Optionen) oder nur das Minimum eingesetzt wird.

Ein äußerst wichtiger Pluspunkt des vorliegenden Verfahrens ist die Tatsache, dass die Sicherheit der hiermit geschützten Daten nicht davon abhängt ob ein Angreifer das Verfahren genau kennt oder nicht.

X. IDEALE AUSGESTALTUNG

Es steht völlig außer Frage: Die ideale Ausgestaltung des Verfahrens bedingt die Timer-Hardwareerweiterung. Da diese hardwareseitig und damit manipulationssicher alle wesentlichen Risiken auszuschließen vermag, steigt die Sicherheit (und in vielen Situationen auch die Performance) wenn diese so viele Aufgaben wie möglich übernimmt. Da sie selbst Schlüssel erzeugt und diese sicher verwahrt, können diese nicht entwendet werden. Wichtig dabei ist jedoch, dass es nur dem Timer-Programm gestattet ist, die De-/ Chiffrierungsdienste der Timer-Hardwareerweiterung in Anspruch zu nehmen. Dies erfolgt über die dargestellten Kontrollmechanismen der Timer-Hardwareerweiterung und der arithmetisch bearbeiteten Übertragung (siehe u.a. D.6).

Die Timer-Hardwareerweiterung kann auch die wechselnden öffentlichen Schlüssel zur Übertragung der Daten in den Aufgaben-Puffer (C.6) erzeugen und den zur Entschlüsselung dieser Daten notwendigen privaten Schlüssel für sich geheim behalten. So würde selbst das Timer-Programm die Daten nie in Klartext „sehen" sondern diese nur an die Timer-Hardwareerweiterung weiter reichen, welche diese dann erst asymmetrische entschlüsselt und

dann symmetrisch mit Masterkey und Zeitschlüssel verschlüsselt. In umgekehrter Datenfluss-Richtung erfolgt das ganze umgekehrt wobei dafür dann der Empfänger einen öffentlichen Schlüssel bereitstellen muss.

Wenn die Performance ausreicht, kann die Übertragung aller Daten, bis auf das unter G.3 dargestellte Niveau für Passwörter ausgebaut werden.

II. Patentansprüche

1. Verfahren zur mehrschichtig geschützten Sicherung von Daten, insbesondere Anmeldedaten und Passwörtern, wobei
a) aus zur Verschlüsselung von Daten eingesetzte Schlüssel (Ausgangsschlüssel) unter Verwendung eines Zeitcodes, der aus einem in Relation zur aktuellen Systemzeit in der Zukunft liegenden System- oder Alarm-Zeitwert (Zeit X) errechnet wird, jeweils ein so genannter Futurekey berechnet wird, und
b) anschließend der/die Ausgangsschlüssel gelöscht werden, und
c) ein Timer so gesetzt (programmiert und gestartet) wird, dass er zur Zeit X abläuft oder auslöst und damit ein Timer-Programm aufruft, welches aus der dann vorliegenden System- oder Timer-Alarm-Zeit den Zeitcode aus a) wieder generiert und damit aus den Futurekeys die in a) verwendeten Ausgangsschlüssel wieder berechnet, um damit anstehende Dechiffrierungs- und Chiffrierungsaufgaben auszuführen;
d) wobei sich das von a) bis c) beschriebene Verfahren kontinuierlich wiederholt und die jeweilige Zeit X, zu der es jeweils möglich ist die Ausgangsschlüssel aus den jeweiligen Futurekeys zu errechnen, stets direkt in einen Timer programmiert wird und gespeicherte Informationen über diese Zeit X inklusive dem Zeitcode aus allen Speichermedien außer dem Timer gelöscht werden;
e) wobei Dechiffrierungs- und Chiffrierungsaufgaben bis zur jeweils nächsten Zeit X zwischengespeichert werden und diese Aufgaben zur Zeit X von dem o.g. Timer-Programm abgearbeitet werden. (Fig. 1)

2. Verfahren nach Anspruch 1, wobei
der verwendete Timer aus Verfahrensschritt c) bis d) des Anspruch 1 auf einer zusätzlichen Hardware, der Timer-Hardwareerweiterung, untergebracht ist und nach dem Starten des Timers bis zu seinem Ablauf/Auslösen gar nicht und danach nur jeweils einmal gelesen werden kann.

3. Verfahren nach einem der vorhergehenden Ansprüche, wobei
jegliche kryptografische Schlüssel, insbesondere im Verfahrensschritt c) des Anspruch 1 wieder errechnete Ausgangsschlüssel, wenn Sie im Arbeitsspeicher, oder einem anderen

Speicher, gespeichert werden, wie dies während deren Verwendung für Dechiffrierungs- und Chiffrierungsaufgaben des Timer-Programms der Fall sein könnte, entweder in mehreren Teilen aufgesplittet in einem Speicherbereich aus Zufallszahlen, der Zufallsmatrix, versteckt gehalten werden, wobei die Zeiger darauf ausschließlich in Prozessorregistern abgelegt werden oder die Schlüssel selbst werden ausschließlich in Prozessorregistern gespeichert; wobei der Speicherbereich für die Zufallsmatrix jedes Mal bevor darin Schlüssel versteckt werden mit neuen Zufallszahlen zu füllen ist.

4. Verfahren nach einem der vorhergehenden Ansprüche, wobei
zum Schutz von Daten diese nacheinander mit mindestens 2 unterschiedlichen Ausgangsschlüsseln mit jeweils unterschiedlichen Verschlüsselungsverfahren verschlüsselt werden, wobei
a) einer dieser Ausgangsschlüsseln, der Masterkey, jedes Mal bevor er zur Verschlüsselung von Daten eingesetzt wird, so bearbeitet wird, dass dies für die Entschlüsselung wieder rekonstruiert werden kann und dennoch möglichst jede Verschlüsselung mit einem etwas anderen Masterkey erfolgt und
b) ein anderer Ausgangsschlüssel, der Zeitschlüssel, in bestimmten Zeitabständen immer wieder neu erzeugt wird, wobei dann jeweils alle damit verschlüsselten Daten mit dem bisherigen Zeitschlüssel entschlüsselt und sofort mit dem neu erzeugten Zeitschlüssel wieder verschlüsselt werden, sodass rotierende Geheimtexte entstehen. (Fig. 3)

5. Verfahren nach einem der vorhergehenden Ansprüche, wobei
zum erweiterten Schutz von Passwörtern und/oder Anmeldenamen, diese bereits bei oder unmittelbar nach der Eingabe, mit einem kryptologischen Einweg-Hashing in Hash-Werte gewandelt werden , welche ganz oder teilweise miteinander kombiniert und zum Teil verkürzt / komprimiert werden sodass eine Rückrechnung unmöglich ist und während der Übermittlung, zusätzlich zu einer etwaigen Transportverschlüsselung wie z.B. bei HTTPS, durch mindestens einen weiteren Geheimschlüssel und mindestens einen vom Empfänger des Passworts übersandten Code verschlüsselt werden, wobei der auf beiden Seiten (Client und Server) bekannte Passwort-Hash und/oder Anmeldename wesentlich zur Generierung der Geheimschlüssel und/oder Codes mit herangezogen wird. (Fig. 4)

6. Verfahren nach einem der vorhergehenden Ansprüche, wobei
zur Verschleierung von Eingaben der Anwender vom Computer Anweisungen erhält wie er die bevorstehende Eingabe abzuwandeln hat.

7. Verfahren nach einem der vorhergehenden Ansprüche, wobei
Daten, insbesondere aber kryptografische Schlüssel mit einem, in bestimmten Zeitabständen zu erneuernden, Schlüssel (Zeitschlüssel) verschlüsselt und dann in mehreren Teilen aufgesplittet auf einem (mit Zufallswerten gefüllten) Speichermedium verteilt gespeichert werden, wobei der maximale Zeitabstand zur Erneuerung des Zeitschlüssels und/oder Schlüssels und/oder deren Neu-Verschlüsselung, die Größe des Speichermediums und dessen Lesegeschwindigkeit so im Verhältnis stehen, dass gilt: Größe in MByte >= Lesegeschwindigkeit in MByte/Sek. * Vielfaches * maximaler Zeitabstand zur Erneuerung/Neuverschlüsselung des Schlüssels in Sekunden.

8. Verfahren nach einem der vorhergehenden Ansprüche, wobei
die sicherheitsrelevanten Programme (u.a. Software zur Chiffrierung und Dechiffrierung / o.g. Timer-Programm) und Daten auf einem nur per manuellem Hardwareeingriff änderbaren/beschreibbaren nichtflüchtigen Speicherchip (z.B. EEPROM auf der o.g. Timer-Hardwareerweiterung) untergebracht sind und entweder sich dieser Speicherchip direkt in einem bestimmten Speicherbereich des Arbeitsspeicher des Systems einblendet oder die o.g. sicherheitsrelevanten Programme und Daten daraus geladen und regelmäßig damit verglichen werden.

9. Verfahren nach einem der vorhergehenden Ansprüche, wobei
die o.g. Timer-Hardwareerweiterung oder eine andere Hardwareeinrichtung prüft, ob der zum Aufruf des Timer-Programms zuständige Interrupt-Zeiger manipuliert wurde, indem die Hardware kurz nach dem Auslösen des Timers eigenständig überprüft, ob der zuständige Prozessor auch tatsächlich in dem Speicherbereich arbeitet in dem das Timer-Programm gespeichert ist und andernfalls System-Alarm auslöst und/oder das System anhält.

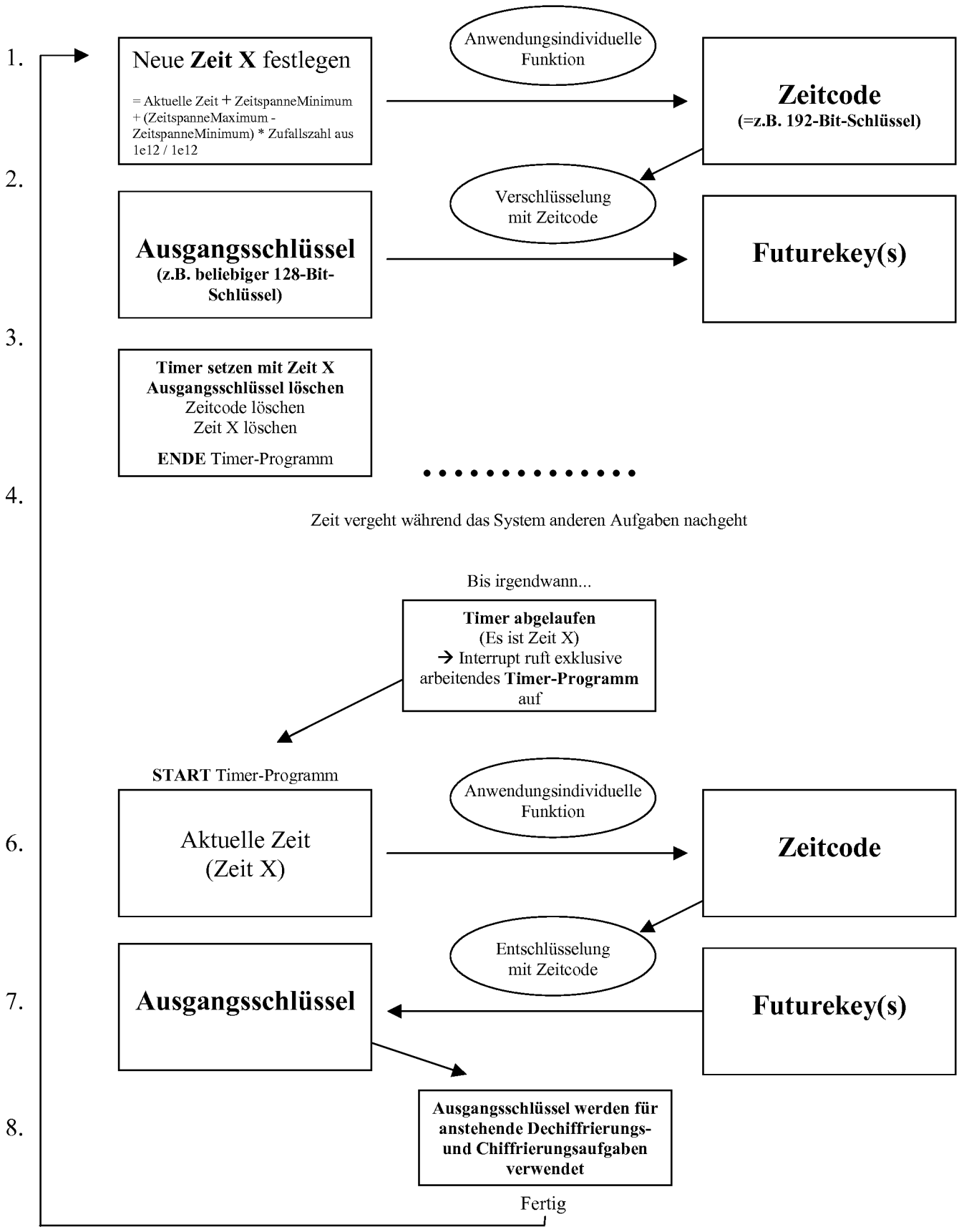
10. Verfahren nach einem der vorhergehenden Ansprüche, wobei
Speicher nur über einen Mikrocontroller / Speicher-Manager ansprechbar ist und dieser

sicherstellt dass zufallsbestimmt ein Teil der Lese-Zugriffe auf diesen Speicher verzögert ausgeführt / beantwortet oder auch abgelehnt werden, außer es handelt sich um Lese-Zugriffe auf bestimmte Zellen dieses Speichers.

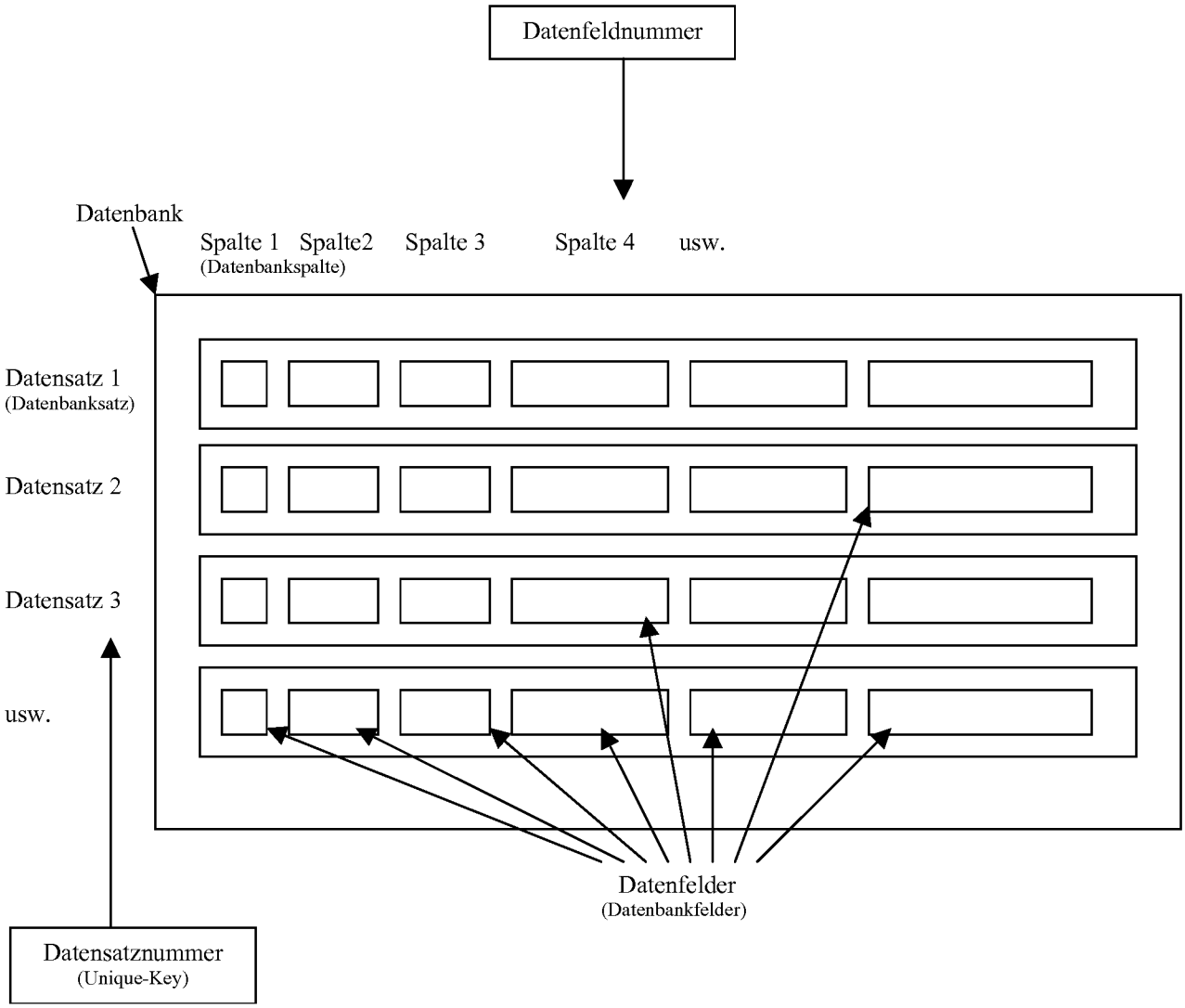
11. Verfahren nach einem der vorhergehenden Ansprüche, wobei
aus Passwörtern und/oder Anmeldenamen und/oder dessen Teilen und/oder Hash-Werten je ein zusätzlicher Einweg-Hashwert gebildet wird, der nicht dauerhaft gespeichert wird, sondern nur dazu dient, den zu dem jeweiligen Anmeldenamen / Passwort gehörenden Datensatz individuell zu verschlüsseln bzw. zu entschlüsseln.

IV. Zeichnungen

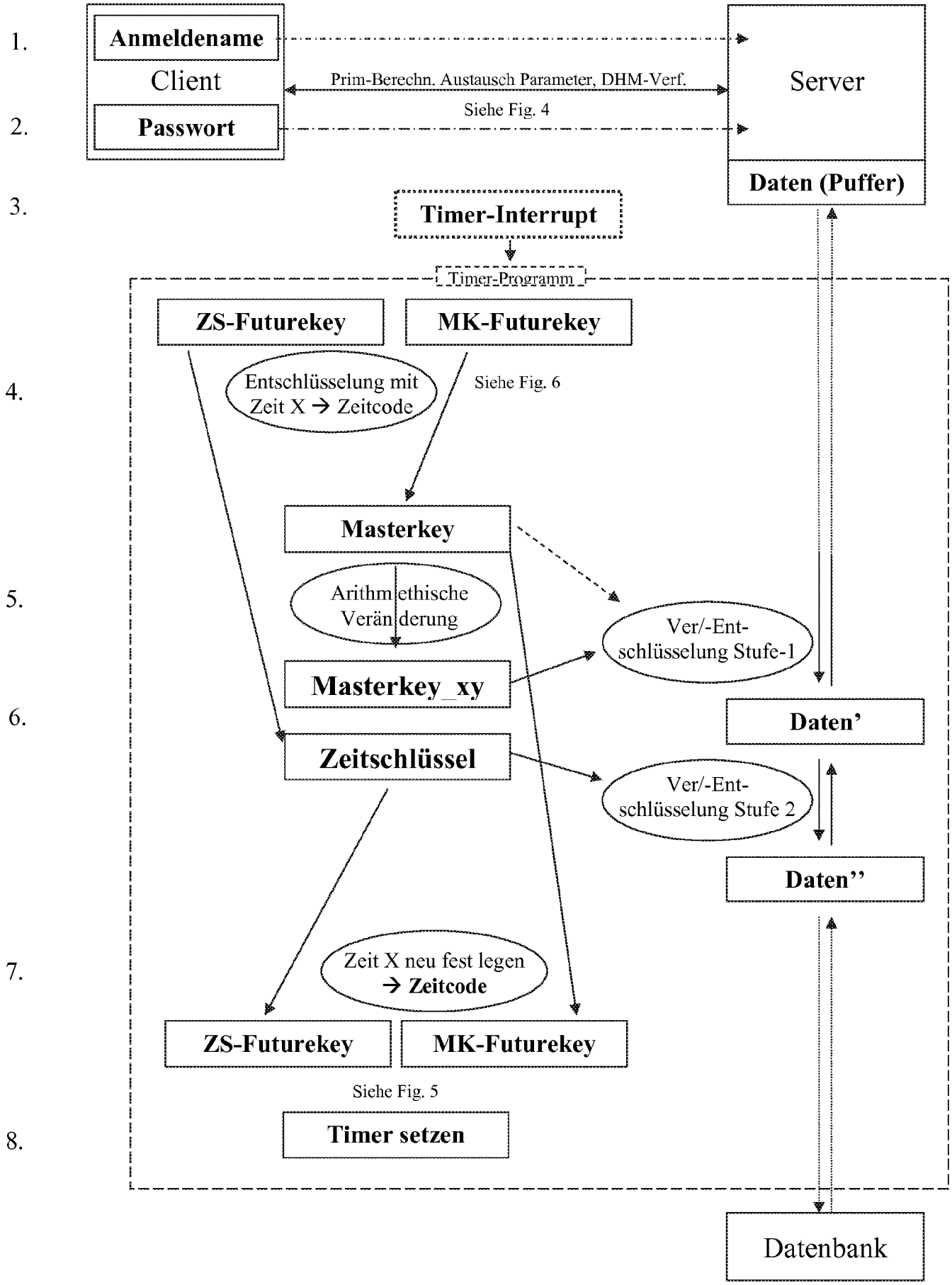
1. Figur 1 - Kern - Ghosten



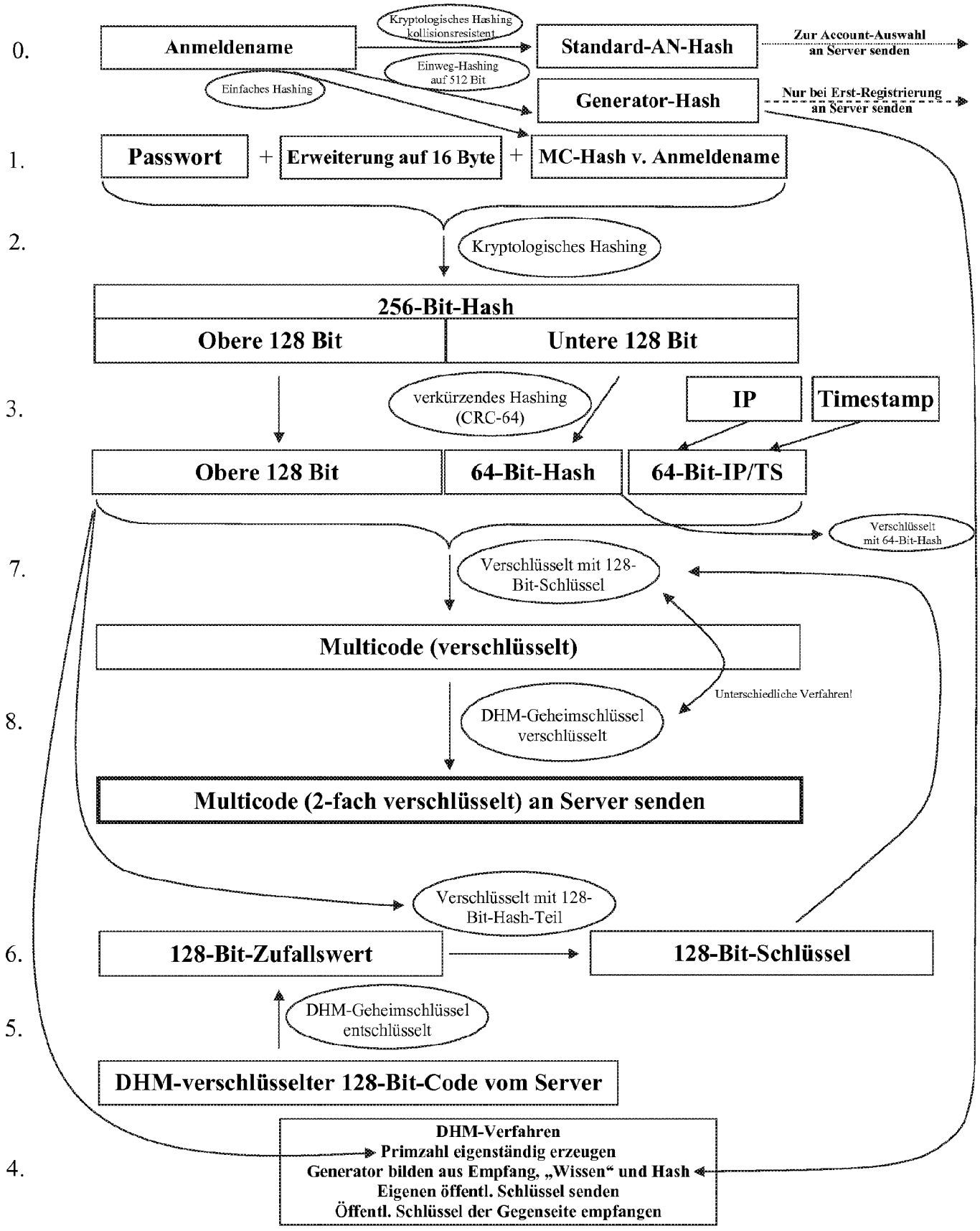
2. Figur 2 - Datenbank



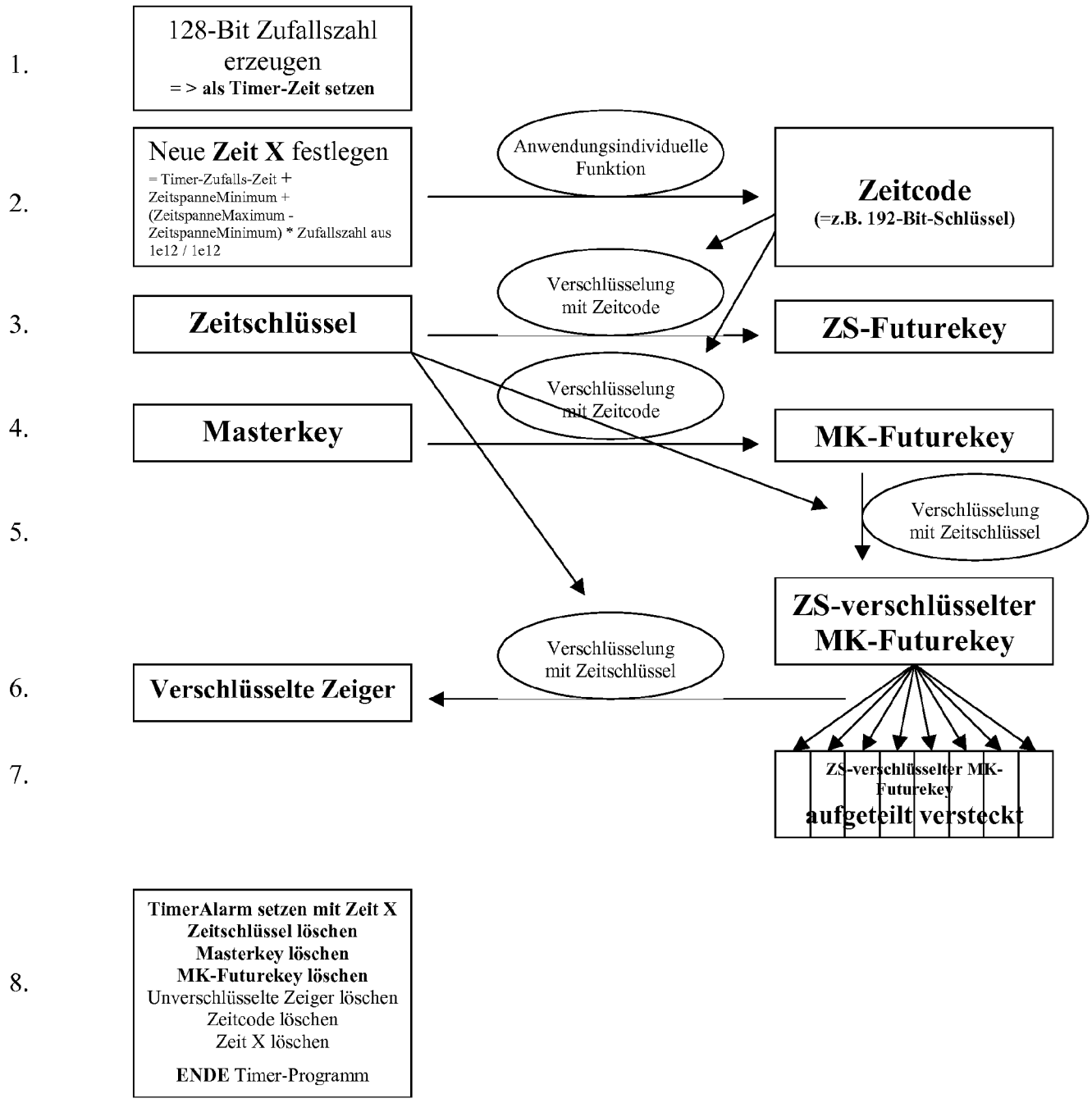
3. Figur 3 - Gesamt-Überblick



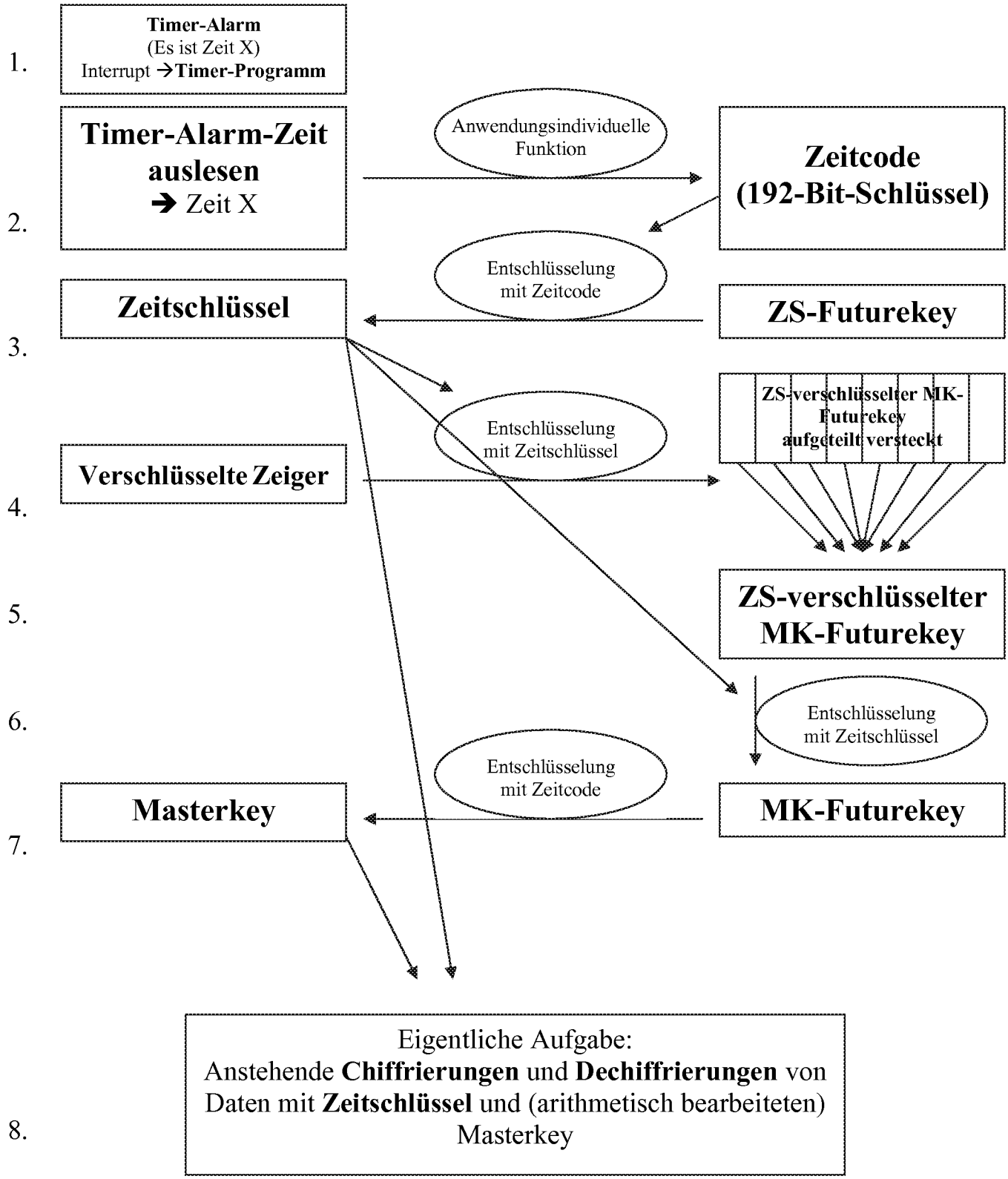
4. Figur 4 - Sichere Passwort-Übermittlung



5. Figur 5 - Ghosten inkl. MK-FK-Versteck

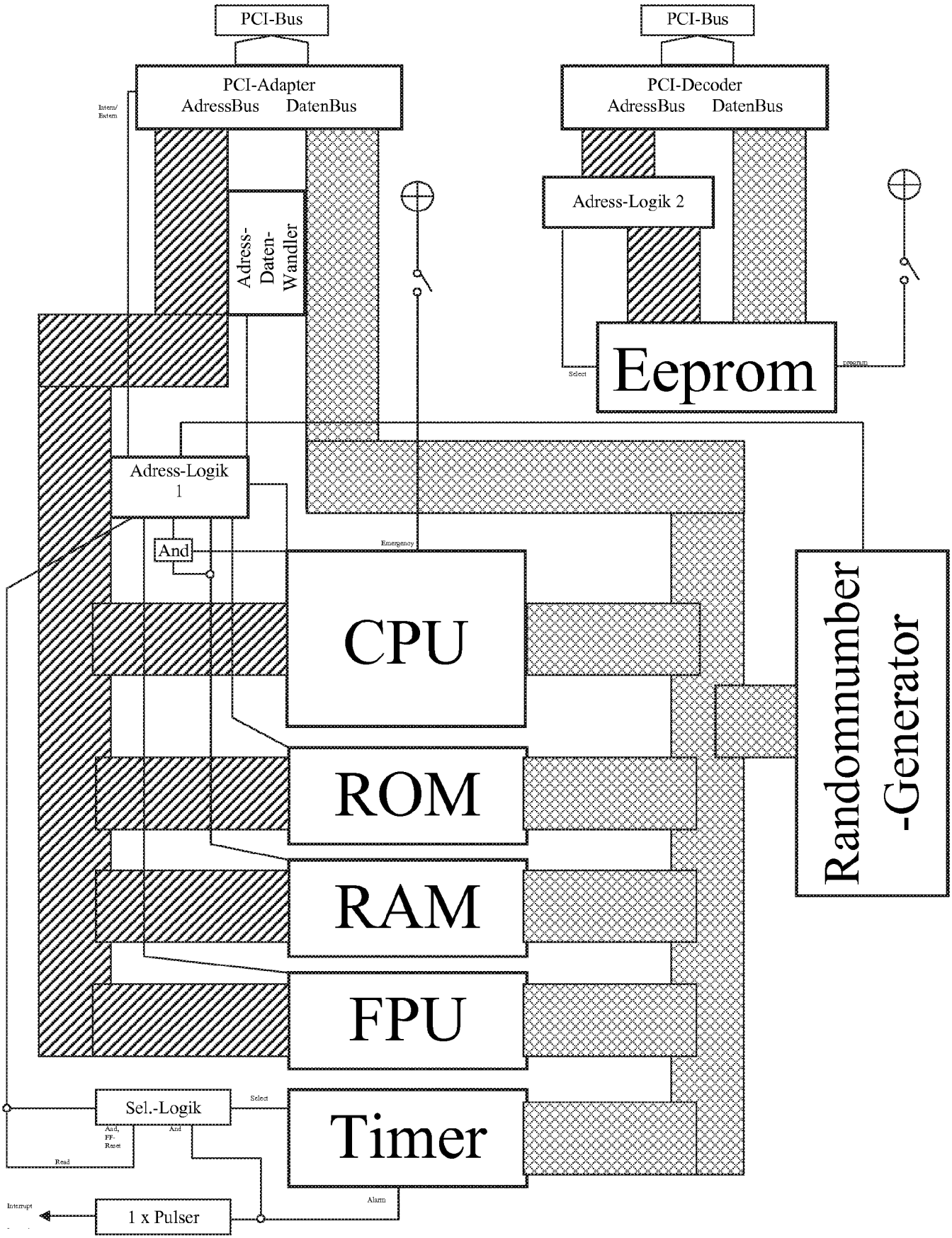


6. Figur 6 - Entghosten inkl. MK-FK-Versteck



Wenn fertig, weiter mit
Pos.1 Fig. 5

7. Figur 7 - Timer-Hardwareerweiterung



8. Figur 8/1 - Siegel

Figur 8/2 - Siegel 2

INTERNATIONAL SEARCH REPORT

International application No
PCT/DE2017/200002

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/31 G06F21/55 G06F21/60 G06F21/62 G06F21/71
G06F21/85 H04L9/00 H04L29/06 H04L9/08
ADD.
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2011/038477 A1 (BILODI PRAKASH B [US]) 17 February 2011 (2011-02-17) figure 4 figure 5	1-11
A	US 8 429 420 B1 (MELVIN STEPHEN WALLER [US]) 23 April 2013 (2013-04-23) figure 4B column 9, line 34 - line 47	1-11

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 April 2017	20/04/2017

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Meis, Marc

3

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Patrick Mcgregor ET AL: "Braving the Cold: New Methods for Preventing Cold Boot Attacks on Encryption Keys", Black Hat USA 2008 Briefings, 7 August 2008 (2008-08-07), XP055106983, Retrieved from the Internet: URL:http://www.crazylazy.info/cons/bh08/attach/BH_US_08_McGregor_Cold_Boot_Attacks.pdf [retrieved on 2014-03-11] page 18 - page 18 -----	3,7

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/DE2017/200002

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011038477 A1	17-02-2011	NONE	
US 8429420 B1	23-04-2013	NONE	

INTERNATIONALER RECHERCHENBERICHT

Internationales Aktenzeichen
PCT/DE2017/200002

A. KLASSIFIZIERUNG DES ANMELDUNGSGEGENSTANDES
INV. G06F21/31 G06F21/55 G06F21/60 G06F21/62 G06F21/71
G06F21/85 H04L9/00 H04L29/06 H04L9/08
ADD.
Nach der Internationalen Patentklassifikation (IPC) oder nach der nationalen Klassifikation und der IPC

B. RECHERCHIERTE GEBIETE

Recherchierter Mindestprüfstoff (Klassifikationssystem und Klassifikationssymbole)
G06F H04L

Recherchierte, aber nicht zum Mindestprüfstoff gehörende Veröffentlichungen, soweit diese unter die recherchierten Gebiete fallen

Während der internationalen Recherche konsultierte elektronische Datenbank (Name der Datenbank und evtl. verwendete Suchbegriffe)
EPO-Internal, WPI Data

C. ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie*	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
A	US 2011/038477 A1 (BILODI PRAKASH B [US]) 17. Februar 2011 (2011-02-17) Abbildung 4 Abbildung 5	1-11
A	US 8 429 420 B1 (MELVIN STEPHEN WALLER [US]) 23. April 2013 (2013-04-23) Abbildung 4B Spalte 9, Zeile 34 - Zeile 47	1-11

-/--

[X] Weitere Veröffentlichungen sind der Fortsetzung von Feld C zu entnehmen [X] Siehe Anhang Patentfamilie

* Besondere Kategorien von angegebenen Veröffentlichungen :
"A" Veröffentlichung, die den allgemeinen Stand der Technik definiert, aber nicht als besonders bedeutsam anzusehen ist
"E" frühere Anmeldung oder Patent, die bzw. das jedoch erst am oder nach dem internationalen Anmeldedatum veröffentlicht worden ist
"L" Veröffentlichung, die geeignet ist, einen Prioritätsanspruch zweifelhaft erscheinen zu lassen, oder durch die das Veröffentlichungsdatum einer anderen im Recherchenbericht genannten Veröffentlichung belegt werden soll oder die aus einem anderen besonderen Grund angegeben ist (wie ausgeführt)
"O" Veröffentlichung, die sich auf eine mündliche Offenbarung, eine Benutzung, eine Ausstellung oder andere Maßnahmen bezieht
"P" Veröffentlichung, die vor dem internationalen Anmeldedatum, aber nach dem beanspruchten Prioritätsdatum veröffentlicht worden ist
"T" Spätere Veröffentlichung, die nach dem internationalen Anmeldedatum oder dem Prioritätsdatum veröffentlicht worden ist und mit der Anmeldung nicht kollidiert, sondern nur zum Verständnis des der Erfindung zugrundeliegenden Prinzips oder der ihr zugrundeliegenden Theorie angegeben ist
"X" Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann allein aufgrund dieser Veröffentlichung nicht als neu oder auf erfinderischer Tätigkeit beruhend betrachtet werden
"Y" Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann nicht als auf erfinderischer Tätigkeit beruhend betrachtet werden, wenn die Veröffentlichung mit einer oder mehreren Veröffentlichungen dieser Kategorie in Verbindung gebracht wird und diese Verbindung für einen Fachmann naheliegend ist
"&" Veröffentlichung, die Mitglied derselben Patentfamilie ist

Datum des Abschlusses der internationalen Recherche	Absendedatum des internationalen Recherchenberichts
10. April 2017	20/04/2017

Name und Postanschrift der Internationalen Recherchenbehörde	Bevollmächtigter Bediensteter
Europäisches Patentamt, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Meis, Marc

3

C. (Fortsetzung) ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie*	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
A	Patrick Mcgregor ET AL: "Braving the Cold: New Methods for Preventing Cold Boot Attacks on Encryption Keys", Black Hat USA 2008 Briefings, 7. August 2008 (2008-08-07), XP055106983, Gefunden im Internet: URL:http://www.crazylazy.info/cons/bh08/attach/BH_US_08_McGregor_Cold_Boot_Attacks.pdf [gefunden am 2014-03-11] Seite 18 - Seite 18 -----	3,7

INTERNATIONALER RECHERCHENBERICHT

Angaben zu Veröffentlichungen, die zur selben Patentfamilie gehören

Internationales Aktenzeichen
PCT/DE2017/200002

Im Recherchenbericht angeführtes Patentdokument	Datum der Veröffentlichung	Mitglied(er) der Patentfamilie	Datum der Veröffentlichung
US 2011038477 A1	17-02-2011	KEINE	
US 8429420 B1	23-04-2013	KEINE	