Abschlußbericht des Projektes

„Bundesweite Einführung eines einheitlichen Record Linkage–Verfahrens in den Krebsregistern der Bundesländer nach dem KRG“

Gefördert von der Deutschen Krebshilfe
(Antragsnummer 70–2043–Ap I)

Dipl.–Inform. Holger Hinrichs, OFFIS

Oldenburg, Juli 1997 — Juni 1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Generierung von Kontrollnummern

Im April 1996 hat die Arbeitsgemeinschaft Leitender Medizinalbeamter (AGLMB) im Rahmen der Umsetzung des Bundeskrebsregistergesetzes (KRG) [Bun94] die Einführung eines einheitlichen Verfahrens zur Generierung von Kontrollnummern in allen Landeskrebsregistern beschlossen. Mit der Umsetzung dieses Beschlusses wurde OFFIS beauftragt. Bei der Deutschen Krebshilfe wurde daraufhin ein Antrag zur finanziellen Förderung dieses Projektes gestellt (siehe [App96]), der im Juni 1997 bewilligt wurde. Das Projekt trägt den Arbeitstitel UNICON (Uniform Control Number Generator) und hat eine Laufzeit von zwei Jahren (Juli 1997 bis Juni 1999).

Das KRG legt fest, daß Datensätze aus unterschiedlichen Meldequellen nicht im Klartext miteinander abgeglichen werden dürfen. Stattdessen muß der Abgleich auf Basis bereits anonymisierter Daten erfolgen. In epidemiologischen Krebsregistern wäre demnach ein Abgleich über die Chiffrate der personenidentifizierenden Daten der einzelnen Meldungen denkbar. Dieser Ansatz ist jedoch nicht sinnvoll, da bereits kleinste Abweichungen in den Klartexten (z. B. „Johannes“ statt „Hannes“) aufgrund der hohen Streuung des Verschlüsselungsverfahrens zu vollkommen unterschiedlichen Chiffraten führen [TA95].

Das KRG sieht daher einen Abgleich anhand von Kontrollnummern vor. Dieses von Arbeitsgruppen in Mainz und Oldenburg entwickelte und evaluierte Konzept, welches ausführlich in [TA95] dargestellt ist, entspricht dem Vorgehen gemäß KRG (siehe [Bun94]). Eine Empfehlung zur technischen Umsetzung der Verfahrensweisen gemäß KRG ist in [AMST96] enthalten und wird in dieser Form auch vom Bundesamt für Sicherheit in der Informationstechnik (BSI) akzeptiert. In dem vorliegenden Abschlußbericht werden die Inhalte des KRG und der Empfehlung zur technischen Umsetzung als bekannt vorausgesetzt.

Bei den Kontrollnummern handelt es sich um deterministische Verschlüsselungen von zuvor standardisierten Komponenten der personenidentifizierenden Daten einer Meldung. Konkret gehen Name, Vorname, Geburtsname, früherer Name, Titel (soweit vorhanden) und Geburtstag ein. Kontrollnummern werden in der Vertrauensstelle parallel zur Verschlüsselung der personenidentifizierenden Daten einer Meldung erzeugt und zusammen mit dem Schlüsseltext und

den epidemiologischen Daten an die Registerstelle übertragen. Kontrollnummern eignen sich für einen Abgleich von Mehrfachmeldungen, da sie zum einen auf standardisierten Klartexten beruhen und zum anderen aus zahlreichen Komponenten bestehen, die separat verschlüsselt sind. So wären zwar in obigem Beispiel die Kontrollnummern zum Vornamen unterschiedlich, bei Übereinstimmung aller anderen Attributwerte könnte jedoch trotzdem eine Zuordnung der Meldungen erfolgen.

Die Generierung von Kontrollnummern besteht aus einer *Standardisierungsphase* und einer *Chiffrierungsphase* [AMST96]. Beide Phasen werden im folgenden kurz erläutert.

1.1.1 Standardisierungsphase

Um Probleme mit unterschiedlichen Darstellungen von Umlauten zu vermeiden, werden Umlaute und ß in die zweibuchstabige Schreibweise umgesetzt (ß → ss, ä → ae, ö → oe, ü → ue), und alles wird einheitlich groß geschrieben. Name, Vorname, Geburtsname und früherer Name werden dann jeweils anhand von Trennzeichen (Blank, Komma, Punkt etc.) in maximal drei Komponenten zerlegt. Die Zerlegung der Namen vereinfacht Zuordnungen zu anderen Meldungen, z. B. wenn nicht immer alle Vornamen angegeben sind, von einem Doppelnamen nicht alle Teile vorliegen oder Namenserweiterungen („von“, „zu“ etc.) vorkommen. Namenserweiterungen sowie vierte und weitere Namensteile werden an die dritte Komponente in der Reihenfolge ihres Auftretens angehängt. Mit Titeln („Prof.“, „Dr.“ etc.) wird analog verfahren.

Um Schreib– und Hörfehler bei Namensangaben zu kompensieren, werden *phonetische Codes* (gemäß Kölner Phonetik [Pos69]) gebildet. Als Eingabe für die Phonetikfunktion dient jeweils die Konkatenation der zuvor standardisierten Namenskomponenten.

Der sog. *DDR–Namenscode* wird aufgenommen, um Abgleiche mit dem Altbestand des Gemeinsamen Krebsregisters der neuen Bundesländer und Berlin (ehemaliges DDR–Krebsregister) zu ermöglichen, in dem zum Teil dieser Code anstelle von Name und Vorname gespeichert ist. Der DDR–Namenscode ordnet einen Namen einer von 100 etwa gleich großen Klassen zu; er besteht aus vier Ziffern, wobei die ersten beiden für den Namen und die nächsten beiden für den Vornamen stehen. Die Zuordnung entspricht genau dem in [Kol75] beschriebenen Schlüssel des Statistischen Bundesamts.

Das Epidemiologische Krebsregister Baden–Württemberg darf laut Landesgesetz ausschließlich zwei spezielle Kontrollnummern verwenden, die aus einer Kombination von Teilen des Vornamens, des Namens bzw. Geburtsnamens sowie dem vollständigen Geburtsdatum bestehen. Diese beiden Kontrollnummern werden aufgenommen, damit bundesweit Abgleiche mit Meldungen aus Baden–Württemberg möglich sind.

Insgesamt werden pro Meldung 22 Komponenten erzeugt (siehe Abb. 1.1). Von diesen sind typischerweise nur ca. 50 % ausgefüllt, da Titel sowie zweite und dritte Namenskomponenten i. allg. fehlen.

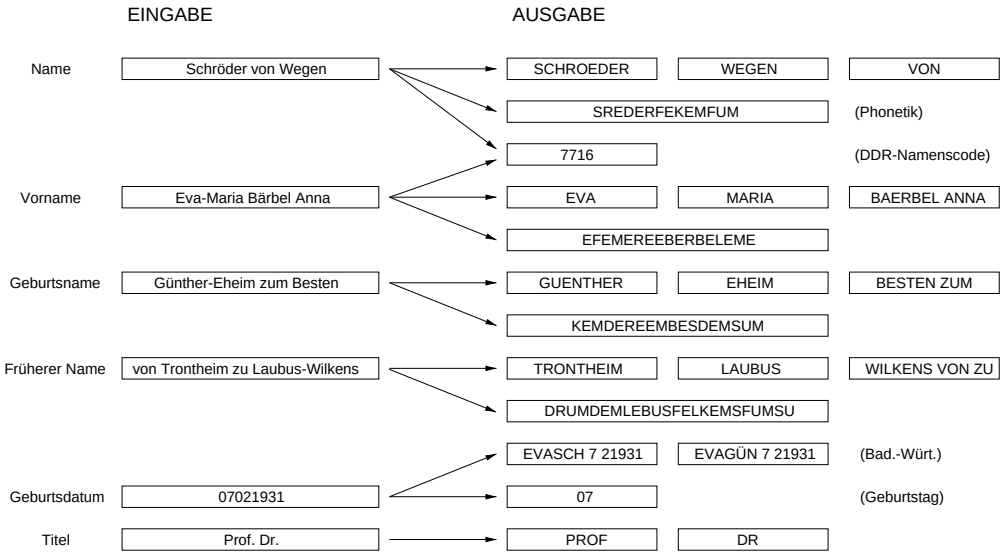

Abbildung 1.1: Generierung von Kontrollnummern – Standardisierungsphase

1.1.2 Chiffrierungsphase

Auf jede der 22 in der Standardisierungsphase erzeugten Komponenten wird nacheinander ein bundesweit einheitliches Einwegverschlüsselungsverfahren (empfohlen: MD5) und ein symmetrisches Chiffrierverfahren (empfohlen: IDEA) angewandt [Sch94]. Als Ergebnis erhält man 22 Kontrollnummern, die nun für Meldungsabgleiche zur Verfügung stehen. Kontrollnummern lassen nicht auf die Identität des einzelnen Patienten schließen. Das symmetrische Chiffrierverfahren unterbindet dabei die Durchführung sog. *Probeverschlüsselungen*. Darunter versteht man die Verschlüsselung verschiedener Klartexte, z. B. häufiger Namen, und den anschließenden Vergleich mit einem gegebenen Chiffrat. Eine Übereinstimmung kommt einer Entschlüsselung des Chiffrats gleich.

1.2 Projektziele

In [AMST96] wurden bereits Leitlinien zur Bildung von Kontrollnummern festgelegt. Dabei konnten jedoch nicht alle Landeskrebsregister berücksichtigt werden, so daß für eine bundesweit einheitliche Lösung kleinere Modifikationen an den Leitlinien vorzunehmen waren.

Die Bereitstellung eines Software–Werkzeugs zur einheitlichen Kontrollnummergenerierung erforderte zunächst die Ermittlung der Hard– und Software–Umgebungen der einzelnen Krebsregister. Zu diesem Zweck hat OFFIS im März 1996 einen entsprechenden Fragebogen verschickt. Nach Auswertung der Fragebogenaktion konnte mit der Entwicklung der Software begonnen werden. Anschließend mußte das Werkzeug in den einzelnen Landeskrebsregistern eingeführt werden.

Die Erzeugung von Kontrollnummern macht langfristig nur Sinn, wenn diese

auch zum Record Linkage (siehe [Jar89]) hinzugezogen werden. Untersuchungen (siehe [SM94]) haben gezeigt, daß ein stochastisches Record Linkage–Verfahren, das auf einzelnen Attributen basiert, zu besseren Zuordnungen führt als ein Verfahren, das feste Kombinationen von mehreren Attributen verwendet. In den Krebsregistern von Rheinland–Pfalz und Niedersachsen wird schon seit längerem das stochastische Abgleichsystem AutoMatch der amerikanischen Firma MatchWare (siehe [Jar94]) eingesetzt. Für den dauerhaften Einsatz des Systems in den Landeskrebsregistern waren jedoch noch rechtliche Fragen zu klären, die Aspekte wie Wartung und Quellcode–Hinterlegung bei einem Treuhänder betrafen.

Die unterschiedlichen Varianten der Kooperation der einzelnen Landeskrebsregister mit der Firma MatchWare (z. B. Bestimmung eines Generalvertreters) sind zu untersuchen, wobei eine der Varianten schließlich umgesetzt werden muß.

1.3 Meilensteinplan

Die einzelnen Arbeitspakete wurden nach folgendem Zeitplan bearbeitet:

- 3. Quartal 1997
  - Auswertung der Fragebogenaktion
  - Überarbeitung der Leitlinien
  - Evaluierung verschiedener kryptographischer Bibliotheken
- 4. Quartal 1997
  - Entwurf einer Systemarchitektur
  - Festlegung der Schnittstellen
  - Auswahl einer kryptographischen Bibliothek
- 1. Quartal 1998
  - Implementierung der Schnittstellen
  - Test des Systems auf verschiedenen Plattformen
  - Dokumentation des Systems
- 2. Quartal 1998 – 2. Quartal 1999
  - Einführung der Software in den Ländern
  - Klärung rechtlicher Fragen

Dieser Zeitplan konnte im wesentlichen eingehalten werden.

Kapitel 2

Anforderungen an das Werkzeug

2.1 Beteiligung der Krebsregister

An der Festlegung der Anforderungen an UNICON waren die Landeskrebsregister maßgeblich beteiligt. So konnten sie sich bei Fragen zur Systemspezifikation einbringen sowie Informationen über die eigene Arbeitsumgebung liefern. Dazu gehörten neben Angaben über die jeweilige Hard– und Softwareausstattung auch die Entscheidung über die Verwendung der Datei– oder Funktionsschnittstelle (siehe Kapitel 3) sowie die Abstimmung von Terminen zur Einführung des Werkzeugs.

2.2 Funktionale Anforderungen

Die Überarbeitung der Leitlinien zur Generierung von Kontrollnummern [AMST96] hat folgende Änderungen ergeben:

Als Trennzeichen in Klartexten ist jetzt neben Punkt, Doppelpunkt, Komma, Semikolon, Minus und Blank auch das Apostroph zugelassen. Abgesehen von den deutschen Umlauten und ß sind sprachspezifische Sonderzeichen verboten. Als Zeichensätze sind ISO–8859 (Windows, UNIX) sowie die IBM–Codepages 437 und 850 (DOS, OS/2) erlaubt.

Die Liste der Namenszusätze (Füllsel) wurde erweitert. Es werden jetzt folgende Füllsel erkannt:

```
AL, AM, AN, AUF, AUS, BEN, D, DA, DAS, DE, DEL,
DELA, DEM, DEN, DER, DI, DOS, DU, EL, EN, ET, L,
LA, LE, LOS, MC, O, OP, T, TE, TEN, TENA, TER,
TO, UND, V, VAN, VO, VOM, VON, Y, ZU, ZUM, ZUR
```

Die Liste der Titel wurde ebenfalls ergänzt und enthält nun folgende Elemente:

```
BARON, BARONIN, DENT, DR, FREIFRAU,
```

```
FREIHERR, GRAEFIN, GRAF, JUR, MED, NAT,
PD, PHIL, POL, PROF, RER, SR, VET
```

Die ersten drei Namensteile, die keine Füllsel sind, werden in den drei standardisierten Namenskomponenten abgelegt. Vierte und weitere Namensteile werden jeweils durch ein Blank getrennt an die dritte standardisierte Namenskomponente angehängt. Daran anschließend werden die Füllsel — falls vorhanden — in der Reihenfolge ihres Auftretens gespeichert, ebenfalls durch je ein Blank separiert. Mit dem Verfahren der Kölner Phonetik (Variante von 1975) wird aus den konkatenierten standardisierten Namenskomponenten (Blanks werden entfernt) ein phonetischer Code gebildet.

Falls Titel gefunden wurden, werden die ersten zwei Titelteile in den beiden standardisierten Titelkomponenten abgelegt. Dritte und weitere Titelteile werden jeweils durch ein Blank getrennt an die zweite standardisierte Titelkomponente angehängt.

Ungültige Datumsangaben (z. B. 31.02.1920) werden nicht akzeptiert, d. h. die entsprechenden Kontrollnummern werden nicht gebildet.

Zur Kontrollnummergenerierung müssen auch unbekannte Werte einheitlich repräsentiert werden, um Vergleichbarkeit zu erreichen. Dies betrifft insbesondere das Geburtsdatum. Intern kann jedes Register wie bisher verfahren (z. B. Wert 0 für fehlende Tages– oder Monatsangabe); nur zur Kontrollnummergenerierung ist ggf. eine (temporäre) Konvertierung erforderlich. Anforderungen aus den Krebsregistern von Rheinland–Pfalz und Baden–Württemberg haben zu folgender Festlegung geführt: eine fehlende Tagesangabe muß durch den Wert 15, eine fehlende Monatsangabe durch den Wert 7 repräsentiert werden. Fehlen Tages– und Monatsangabe, ist der 1.7. zu wählen (Mitte des Jahres).

Während die Kontrollnummern 1 bis 20 jeweils mit dem Einwegverfahren MD5 und anschließend mit dem symmetrischen Verfahren IDEA (in der Betriebsart Cipher Feedback) chiffriert werden, werden die Kontrollnummern 21 und 22 (Baden–Württemberg) „nur“ mit IDEA verschlüsselt, da diese bereits einweg–chiffriert vorliegen. Diese Vorgehensweise hat folgenden Hintergrund: Laut [AMST96] sollen die beiden baden–württemberger Kontrollnummern „als Einzelattribute zur Erzeugung von Kontrollnummern für den bundesweiten Abgleich verwendet werden.“ Damit sie in Baden–Württemberg wie bisher weiterverwendet werden können, wird auf eine zusätzliche Einwegverschlüsselung (per MD5) verzichtet.

Für einen bundesweiten Abgleich sind die baden–württemberger Kontrollnummern mit dem bundeseinheitlichen Schlüssel symmetrisch zu chiffrieren. Daraus folgt, daß auch für das Epidemiologische Krebsregister Baden–Württemberg eine — wenn auch abgespeckte — Softwarekomponente zu entwickeln ist. Die anderen Bundesländer speichern baden–württemberger Kontrollnummern in symmetrisch überverschlüsselter Form, so daß diese landesspezifische Verschlüsselung rückgängig gemacht werden muß, bevor eine bundeseinheitliche Verschlüsselung erfolgen kann.

2.3 Technische Anforderungen

2.3.1 Inhalte der Fragebogenaktion

Ziel des Fragebogens war die Ermittlung der unterschiedlichen Systemvoraussetzungen und der speziellen Anforderungen (z. B. Systemintegration in bereits bestehende Strukturen) der einzelnen Landeskrebsregister hinsichtlich des Kontrollnummergenerierungssystems. Auf Basis dieser Erfassung konnten dann Aufwand und zeitlicher Rahmen für die Entwicklung des Systems festgelegt werden. Im folgenden sind die gestellten Fragen im einzelnen aufgeführt.

Hardware

1. Welche Rechnerysteme verwenden Sie (Prozessor, Plattenkapazität, Display, Hauptspeicherkapazität, ...)? Für welche Rechnerysteme soll das Kontrollnummerngenerierungssystem verfügbar sein?

2. Ist für die Kontrollnummerngenerierung ein eigener Rechner vorgesehen?

3. Welche Netzwerkstrukturen sind vorhanden?

4. Soll die Kontrollnummerngenerierung über Netz betrieben werden?

5. Sind in nächster Zeit Veränderungen in Ihrer Hardwareumgebung geplant? Wenn ja, welche?

Software

1. Welche Betriebssysteme verwenden Sie? Für welche Betriebssysteme soll das Kontrollnummerngenerierungssystem verfügbar sein?

2. Welche User-Interface-Management-Systeme verwenden Sie? Für welche User-Interface-Management-Systeme soll das Kontrollnummerngenerierungssystem verfügbar sein?

3. Welche Programmiersprachen verwenden Sie? In welchen Programmiersprachen sollte das Kontrollnummerngenerierungssystem entwickelt werden?

4. Welche Compiler verwenden Sie und in welchen Versionen?

5. Welches Datenmodell benutzen Sie (z.B. relationales, Codasyl, ...)?

6. Welche Datenbank-Managementsysteme (z.B. ORACLE, Informix, ...) verwenden Sie?

7. Wie sind Ihre Relationenschemata für Meldungen bzw. die vorhandenen Dateiformate festgelegt?

8. Sind in nächster Zeit Veränderungen in Ihrer Softwareumgebung geplant? Wenn ja, welche?

Sonstige Anforderungen

1. Beschreiben Sie kurz Ihre bisherige Vorgehensweise der Meldungsbearbeitung und skizzieren Sie, wo und wie die Kontrollnummerngenerierung aus Ihrer Sicht darin integriert werden soll?

2. Welche Inputschnittstellen stellen Sie für die Kontrollnummerngenerierung bereit (z.B. Datenbankschnittstelle, Dateischnittstelle, Fremdsysteme, ...)? Wie sieht das Inputformat im Detail aus?

3. Welche Outputschnittstellen benötigen Sie für die Kontrollnummerngenerierung (z.B. Datenbankschnittstelle, Dateischnittstelle, Fremdsysteme, ...)? Wie sollte das Outputformat gestaltet sein?

4. Muß das Kontrollnummerngenerierungssystem mehrbenutzerfähig sein?

5. In welcher Form soll die Kontrollnummerngenerierung realisiert werden (z.B. Kontrollnummern werden pro Meldung generiert, Kontrollnummern werden für eine Menge von Meldungen (z.B. „über Nacht“) generiert)?

6. Stellen Sie besondere Anforderungen an den Datendurchsatz bei der Kontrollnummerngenerierung (z.B. durch periodisch hohes Datenaufkommen, ...)?

7. Können und wollen Sie das Kontrollnummerngenerierungssystem selbst in Ihre Infrastruktur integrieren?

8. Können Sie uns bei der Entwicklung des Kontrollnummerngenerierungssystems unterstützen? Wenn ja, wie?

9. Sind in nächster Zeit Veränderungen geplant, die sich auf das Kontrollnummerngenerierungssystem auswirken könnten? Wenn ja, welche?

10. Wann wird das Kontrollnummerngenerierungssystem benötigt (terminliche Vorstellung)?

2.3.2 Ergebnisse der Fragebogenaktion

Bis auf Baden–Württemberg, Bremen, Bayern und Hessen haben alle Landeskrebsregister den Fragebogen beantwortet[1]. Die Auswertung der Antworten ergab hardwareseitig ein recht homogenes, softwareseitig jedoch ein sehr heterogenes Bild.

Als Rechnerplattformen werden praktisch in allen Registern i486– oder Pentium–PCs mit ausreichender Hauptspeicher– und Plattenkapazität eingesetzt. Eine Ausnahme stellen die Krebsregister von Rheinland–Pfalz und Niedersachsen dar, wo neben PC–Clients eine IBM RS/6000 bzw. eine SUN Ultra Sparc verwendet werden.

[1] Einige Register haben zum Zeitpunkt der Fragebogenaktion noch nicht existiert bzw. befanden sich gerade im Aufbau. Im Laufe des Projektes konnten jedoch auch deren Anforderungen noch angemessen berücksichtigt werden.

Krebsregister	Hardware	Betriebssysteme
Gemeinsames Krebsregister	PCs (486/Pentium) mit mind. 4MB RAM und 4GB HD	MS–DOS, Windows95, Novell 3.12
Rheinland–Pfalz	PCs (486/Pentium) mit 16MB RAM und 250MB – 1GB HD, IBM RS/6000 mit 64MB RAM und 2GB HD	MS–DOS, Windows 3.11, Windows95, Windows NT, AIX (auf RS/6000)
Münster	2 PC–Server (jeweils Pentium 133 mit 256MB RAM und 2GB HD), 19 PC-Clients (diskless 486 oder Pentium 133 mit 32MB RAM)	Windows 3.1/3.11, Windows95, Novell 4.1
Hamburg	1 PC–Server (486 mit 64MB RAM und 1GB HD), 5 PC–Clients (486 mit 8MB RAM und 200MB HD)	SCO Unix, Windows 3.1 für SCO Unix, Ethernet
Schleswig–Holstein	PC (Pentium 166) mit 32MB RAM und 4GB HD	Windows95, Novell 4.11
Niedersachsen	Solaris–Server (Sun Sparc Ultra), PC–Clients	Sun Solaris
Saarland	PC–Server (Pentium 133 mit 16MB RAM und 1.2GB HD), PC–Client (486 mit 8MB RAM und 200MB HD)	MS–DOS, Novell Personal NetWare

Tabelle 2.1: Antworten auf den Fragebogen, Teil 1

Unter den Betriebssystemen für PC–Plattformen dominieren die Windows 95/NT–Lösungen, aber auch Windows 3.1, MS–DOS, Novell und SCO/UNIX kommen vor.

Viele Krebsregister arbeiten mit PC–Datenbanken wie Clipper oder Paradox. Die darauf basierenden Anwendungen wurden i. a. mit der jeweils zugehörigen 4GL entwickelt. In Hamburg und Niedersachsen kommen Informix bzw. Oracle zum Einsatz.

Als Schnittstellen zur Kontrollnummergenerierungskomponente wünschen sich die Landeskrebsregister teils eine Funktions-, teils eine Datei–Schnittstelle. Dies hängt in erster Linie von der verwendeten Programmiersprache ab (s. o.). Lassen sich z. B. aus dem Quellcode heraus keine C–Routinen aufrufen, so ist eine Kommunikation über Dateien unumgänglich.

Die Tabellen 2.1, 2.2 und 2.3 zeigen die Ergebnisse der Fragebogenaktion im Überblick. Dabei ist zu beachten, daß einige Angaben inzwischen überholt sind. Änderungen der Hard– und Softwareumgebungen, die bis Juni 1999 durchgeführt wurden, konnten noch im UNICON–Projekt berücksichtigt werden.

Krebsregister	Programmiersprachen	Datenbankmanagementsysteme
Gemeinsames Krebsregister	CA Clipper 5.2 bzw. CA VisualObjects	Clipper bzw. CA VisualObjects
Rheinland–Pfalz	Borland C++ 3.1 bzw. 5.0, Delphi 2.0, ObjectPal	Borland Paradox, Ingres (auf RS/6000)
Münster	Gupta SQLWindows 5.0.2 bzw. Centura TeamDeveloper	Gupta SQLBase 5.2.0 bzw. Centura SQLBase
Hamburg	Informix 4GL, SCO C	Informix SE
Schleswig–Holstein	Clarion CW 2.1, Oberon/F 1.21	Pervasive (Btrieve)
Niedersachsen	SC3.0.1 C++	Oracle 7.3.2.1.0
Saarland	Clipper Sommer 87	dBase

Tabelle 2.2: Antworten auf den Fragebogen, Teil 2

Krebsregister	Schnittstelle	Terminvorstellung
Gemeinsames Krebsregister	ASCII–Dateien, Einzelaufrufe	1. Vorabversion in 12/96, 2. Vorabversion in 7/97
Rheinland–Pfalz	Einbindung von C(++)–Funktionen, ggf. ASCII–Dateien, Batchbetrieb	So bald wie möglich
Münster	Einbindung von C(++)–Funktionen, Batchbetrieb	Kein Zeitdruck
Hamburg	DB– oder Datei–Schnittstelle oder Funktionsaufrufe, Einzelaufrufe	Informix SE
Schleswig–Holstein	ASCII–Dateien, Batchbetrieb	Spätsommer 97
Niedersachsen	Einbindung von C(++)–Funktionen	So bald wie möglich
Saarland	ASCII–Dateien	Keine Angabe

Tabelle 2.3: Antworten auf den Fragebogen, Teil 3

Kapitel 3

Entwurf des Werkzeugs

3.1 Systemarchitektur

Das UNICON–Softwaremodul soll an verschiedene Applikationen angebunden werden können. Dabei handelt es sich i. a. um Frontends zur Erfassung von Meldungen in der Vertrauensstelle eines epidemiologischen Krebsregisters gemäß Gesetz über Krebsregister (KRG). Da in den Landeskrebsregistern vorwiegend Eigenentwicklungen eingesetzt werden, stellt sich die Software–Landschaft hier sehr heterogen dar: Nicht nur die Frontends sind unterschiedlich, auch die verwendeten Entwicklungssysteme differieren. So kommen z. T. die 4GLs von PC–Datenbanken wie Clipper oder Paradox, z. T. C++–Compiler mit Anbindung an RDBMS wie ORACLE zum Einsatz. Daraus ergibt sich für das UNICON–System das Problem, eine Schnittstelle zur Verfügung zu stellen, die von allen relevanten Frontends verwendet werden kann. So erlauben einige Programme einen Aufruf von externen C– oder C++–Routinen, andere jedoch nicht.

Den kleinsten gemeinsamen Nenner bildet daher eine Dateischnittstelle: Das Frontend schreibt personenidentifizierende Daten in eine Datei, deren Inhalt anschließend vom UNICON–System ausgelesen und verarbeitet wird. Die von UNICON ebenfalls in eine Datei geschriebenen verschlüsselten Kontrollnummern können dann wieder in das Frontend übernommen werden. Neben der grundlegenden Funktionalität der Generierung von Kontrollnummern bietet die Dateischnittstelle u. a. die Möglichkeit einer Konvertierung zwischen Storage– und Linkage–Format sowie einer RSA–Chiffrierung (siehe [AMST96]). Würde UNICON lediglich diese (relativ restriktive) Dateischnittstelle anbieten, könnten diejenigen Applikationen mit Fähigkeit zum Aufruf externer Routinen die Low–Level–Funktionalität von UNICON nicht nutzen. Aus diesem Grund wird zusätzlich eine C–Funktionsschnittstelle bereitgestellt. Diese besteht aus mehreren Schichten, die unterschiedlichen Abstraktionsebenen entsprechen, wobei die Routinen der unteren Schichten nach oben „durchgeschleift“ werden. Die unterste Schicht kapselt die Funktionalität einer kryptographischen Bibliothek. Diese kann also bei Bedarf ausgetauscht werden, ohne daß die darüberliegenden Schnittstellen davon beeinflußt werden. In der Entwicklungsphase von UNICON wurden verschiedene Kryptobibliotheken evaluiert, insbesondere SecuDE und SSLeay.

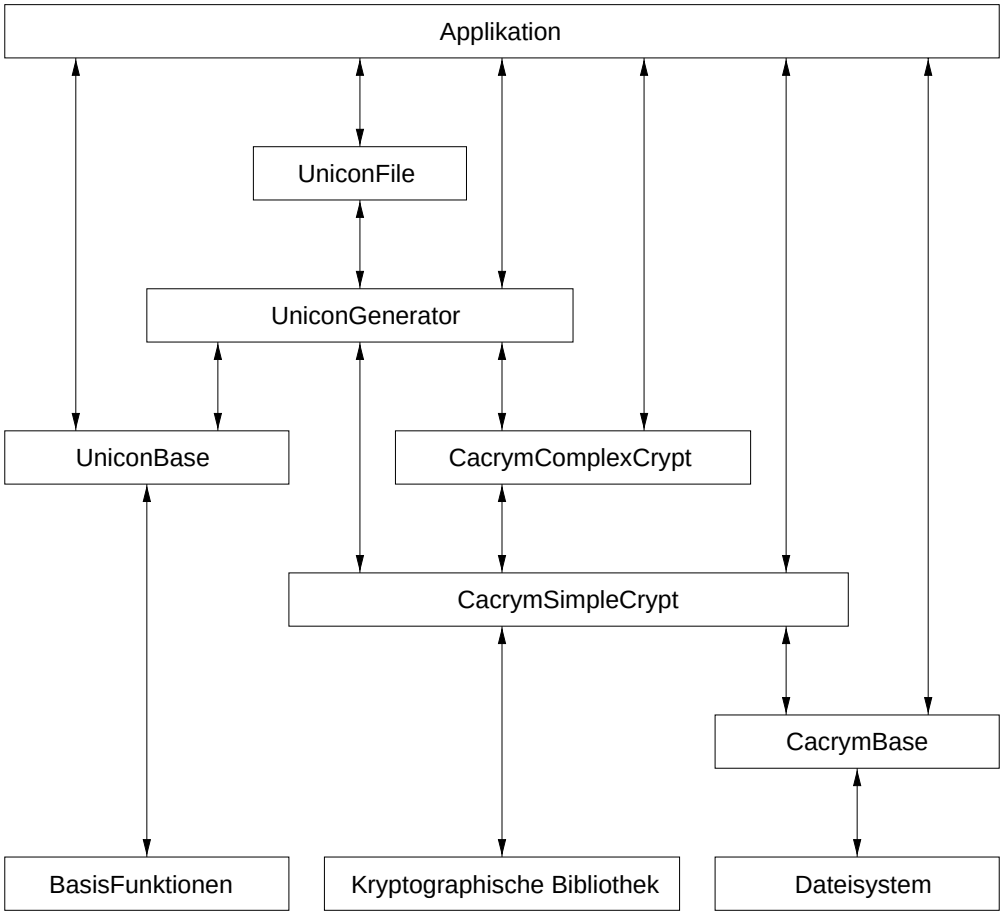

Abbildung 3.1: Die Architektur des UNICON–Systems

Zusammenfassend sieht die UNICON–Architektur bottom–up wie folgt aus (siehe Abb. 3.1): Das Modul `CacrymBase` basiert auf dem Dateisystem und enthält Funktionen zum Schlüsselmanagement sowie zur Formatierung von Zeichenketten. Aufsetzend auf einer kryptographischen Bibliothek und `CacrymBase`, stellt die Low–Level–Kryptographie–Schnittstelle `CacrymSimpleCrypt` eine Basisfunktionalität zur Verfügung, die u. a. kryptographische Verfahren wie MD5, IDEA, oder RSA[1] umfaßt. Auf der nächsthöheren Schicht befindet sich die High–Level–Kryptographie–Schnittstelle `CacrymComplexCrypt`, die Routinen zur Chiffrierung von Kontrollnummern enthält. Sowohl `CacrymSimpleCrypt` als auch `CacrymComplexCrypt` werden von dem Modul `UniconGenerator` verwendet, das aus personenidentifizierenden Daten Kontrollnummern bildet. `UniconGenerator` greift dazu auf Funktionen des Moduls `UniconBase` zurück, welches vorwiegend zur Standardisierung von Eingabedaten dient. `UniconGenerator` wiederum stellt die Basis

[1] Das RSA–Verfahren ist zwar zur Generierung von Kontrollnummern nicht erforderlich, erlaubt es jedoch den Krebsregistern, auch die asymmetrische Verschlüsselung mit der UNICON–Komponente durchzuführen. Es handelt sich sozusagen um ein „Bonus–Feature“.

für die Datei-Schnittstelle `UniconFile` dar. Als Frontend fungiert schließlich die in der Vertrauensstelle des jeweiligen Krebsregisters eingesetzte Applikation, an die das UNICON–System angebunden wurde. Diese Applikation hat aus Gründen der maximalen Flexibilität Zugriff auf alle sechs untergeordneten Schnittstellen. Falls das der Applikation zugrundeliegende Entwicklungssystem eine Verwendung der (komfortablen) Funktionsschnittstellen nicht zuläßt, muß als Notlösung auf die Dateischnittstelle zurückgegriffen werden.

3.2 Die Funktionsschnittstelle

Für alle angebotenen Funktionen (außer die Initialisierungsfunktionen) gilt, daß als erster Parameter ein Handle auf die UNICON–Umgebung übergeben werden muß. In der folgenden Schnittstellenspezifikation ist dieser Handle mit `void* unicon_generator` bezeichnet. Bevor die UNICON–Funktionalität genutzt werden kann, ist einmalig die Methode `UInitGenerator()` aufzurufen, die ein Handle zurückliefert, welches bei den nachfolgenden Funktionsaufrufen anzugeben ist. Schließlich muß die UNICON–Umgebung durch Aufruf von `UFinishGenerator()` wieder freigegeben werden; das Handle ist danach ungültig und darf nicht mehr benutzt werden.

3.2.1 Die Schnittstelle `CacrymBase`

Kryptographische Verfahren arbeiten — solange es sich nicht um Einwegverfahren wie MD5 handelt — mit sog. *Schlüsseln*. Ein Schlüssel kann sowohl zur Chiffrierung als auch zu Dechiffrierung verwendet werden. Bei symmetrischen Verfahren wie IDEA wird derselbe Schlüssel zum Ver– und Entschlüsseln benutzt, bei asymmetrischen Verfahren wie RSA sind Chiffrier– und Dechiffrierschlüssel hingegen unterschiedlich. Allgemein gilt, daß ein Schlüssel dauerhaft gespeichert werden muß, um bei Bedarf darauf zugreifen zu können. In `CACRYM` werden Schlüssel paßwortgeschützt in ASCII–Dateien verwaltet. Die Klasse `CacrymBase` stellt eine Reihe von Methoden zur Verfügung, die der Schlüsselverwaltung dienen, z. B. zum Laden von Schlüsseln aus einer Datei oder Speichern in eine Datei.

Methoden

```
char* UConvertBinToAscii( void* unicon_generator,
                          unsigned char *input,
                          int len );
```

Die Funktion konvertiert die Zeichenkette `input`, die auch nicht–druckbare Zeichen enthalten kann und aus `len` Zeichen besteht, in eine Zeichenkette, die sich ausschließlich aus druckbaren Zeichen zusammensetzt (ASCII–Codes: ...), und liefert diese zurück. Die Länge des Ausgabestrings ist i. a. größer als die von `input` und kann wie üblich mit Funktionen wie `strlen()` abgefragt werden. `UConvertBinToAscii()` und `UConvertAsciiToBin()` (s. u.) werden vorwiegend von den Chiffrier– und Dechiffrierverfahren der abgeleiteten Klassen verwendet,

um z. B. geladene Schlüssel für deren Benutzung oder Chiffrate für deren Weiterverarbeitung vorzubereiten. Der intern verwendete Algorithmus ist eine auf Zeichenketten im Hauptspeicher angepaßte Variante des auf Dateikonvertierung zugeschnittenen btoa– bzw. atob–Verfahrens Version 5.2.1 (Paul Rutter, Joe Orost, Stefan Parmark).

```
int UConvertAsciiToBin( void* unicon_generator,
                        char *input,
                        unsigned char *&output,
                        int &len );
```

Diese Funktion stellt die Umkehrfunktion zu `UConvertBinToAscii()` dar. Die nur aus druckbaren Zeichen bestehende Zeichenkette `input` wird in die Zeichenkette `output` mit der Länge `len` konvertiert, wobei `output` auch nicht–druckbare Zeichen enthalten kann. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int USetKey( void* unicon_generator,
             char *id,
             char *password,
             int type );
```

Die Funktion setzt den Schlüssel mit der Kennung `id`, vorausgesetzt das Paßwort `password` ist für die angegebene Kennung gültig. Desweiteren muß ein Schlüsseltyp angegeben werden, um zwischen IDEA– und RSA–Schlüsseln differenzieren zu können. Erlaubte Ausprägungen sind 0 (RSA Public Key), 1 (RSA Secret Key) und 2 (IDEA Key). Der gesetzte Schlüssel wird in die interne Liste der gesetzten Schlüssel aufgenommen. Im Fehlerfall wird 1 zurückgegeben, sonst 0. Schlüssel können mit den Methoden `UGenerateKeyIDEA()` bzw. `UGenerateKeyRSA()` aus `CacrymSimpleCrypt` erzeugt werden.

```
void UUnsetKey( void* unicon_generator,
                char* id,
                int type );
```

Die Funktion bildet die Umkehrfunktion zu `USetKey()`. Ein zuvor gesetzter Schlüssel mit der Kennung `id` wird aus der Liste der gesetzten Schlüssel entfernt. Wiederum ist dazu ein Schlüsseltyp anzugeben. Erlaubte Ausprägungen sind 0 (RSA Public Key), 1 (RSA Secret Key) und 2 (IDEA Key).

```
int UDeleteKey( void* unicon_generator,
                char* id,
                char* password,
                int type );
```

Die Funktion löscht den Schlüssel mit der Kennung `id` aus der Schlüsseldatei, vorausgesetzt das Paßwort `password` ist für die angegebene Kennung gültig.

Der Schlüssel darf nicht gesetzt sein. Wiederum ist dazu ein Schlüsseltyp anzugeben. Erlaubte Ausprägungen sind 0 (RSA Public Key), 1 (RSA Secret Key) und 2 (IDEA Key). Im Fehlerfall wird 1 zurückgegeben, sonst 0. Achtung: Die Funktion ist mit Vorsicht zu genießen, da Schlüssel *unwiederbringlich* gelöscht werden.

```
int UChangePassword( void* unicon_generator,
                     char* id,
                     char* old_password,
                     char* new_password,
                     int type );
```

Die Funktion ändert das Paßwort des Schlüssels mit der Kennung `id` vom Typ `type` von `old_password` nach `new_password`. Erlaubte Ausprägungen für `type` sind 0 (RSA Public Key), 1 (RSA Secret Key) und 2 (IDEA Key). Im Fehlerfall wird 1 zurückgegeben, sonst 0. Aus Sicherheitsgründen sollten Paßwörter in regelmäßigen Abständen geändert werden.

3.2.2 Die Schnittstelle `CacrymSimpleCrypt`

`CacrymSimpleCrypt` ist Subklasse von `CacrymBase` und stellt eine Reihe kryptographischer Verfahren bereit. Die darunterliegende kryptographische Bibliothek wird gekapselt, so daß ein Austausch der Bibliothek vorgenommen werden kann, ohne daß sich die Schnittstelle ändert.

Basisklasse

`CacrymBase`

Methoden

```
void UEncryptMD5Converted( void* unicon_generator,
                           char* plaintext,
                           char* &hashtext );
```

Die Funktion generiert zur Zeichenkette `plaintext` den Hashwert nach dem MD5–Verfahren und liefert diesen in `hashtext` zurück. Intern wird der Hashwert zuvor noch mittels `UConvertBinToAscii()` aus `CacrymBase` so formatiert, daß er nur noch druckbare Zeichen enthält. Deshalb ist der Hashwert i. a. länger als die für MD5 üblichen 16 Zeichen.

```
int UGenerateKeyIDEA( void* unicon_generator,
                      char* id,
                      char* password,
                      int mode );
```

Die Funktion generiert einen IDEA–Schlüssel für die Betriebsart `mode` mit der Kennung `id`. Dabei kommen die Betriebsarten Electronic Code Book (ECB, Mode 0), Cipher Feed Back (CFB, Mode 1), Output Feed Back (OFB, Mode 2)

und Cipher Block Chaining (CBC, Mode 3) in Frage. Die Betriebsart ECB ist jedoch aufgrund von mangelhafter Sicherheit nicht zu empfehlen. Der Schlüssel wird mit dem Paßwort `password` geschützt und in einer Datei gesichert, es sei denn, die gewünschte Kennung ist bereits vergeben. In diesem Fall wird 1 zurückgegeben, sonst 0. Vor der Speicherung erfolgt eine Formatierung des Schlüssels mittels `UConvertBinToAscii()`.

```
int UEncryptIDEAConverted( void* unicon_generator,
                           char* plaintext,
                           char* id,
                           int mode,
                           char* &ciphertext );
```

Die Funktion verschlüsselt den Klartext `plaintext` mit dem Schlüssel der Kennung `id` unter Verwendung des IDEA–Blockchiffrierverfahrens in der Betriebsart `mode`. Die zulässigen Betriebsarten sind bei `UGenerateKeyIDEA()` aufgeführt (s. o.). Vor der Verwendung des Schlüssels wird dieser intern mit `UConvertAsciiToBin()` in sein ursprüngliches Format zurückgeführt. Als Ergebnis wird der mit `UConvertBinToAscii()` nachbearbeitete Schlüsseltext `ciphertext` zurückgeliefert. Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UDecryptIDEAConverted( void* unicon_generator,
                           char* ciphertext,
                           char* id,
                           int mode,
                           char* &plaintext );
```

Diese Methode stellt die Umkehrfunktion zu `UEncryptIDEAConverted()` dar. Das Chiffrat `ciphertext` wird mit dem Schlüssel der Kennung `id` unter Verwendung des IDEA–Blockchiffrierverfahrens in der Betriebsart `mode` entschlüsselt. Die Betriebsart muß dabei dieselbe sein, die auch bei der Verschlüsselung eingesetzt wurde. Vor der Verwendung des Schlüssels wird dieser intern mit `UConvertAsciiToBin()` in sein ursprüngliches Format zurückgeführt; gleiches gilt für das Chiffrat. Als Ergebnis wird der Klartext `plaintext` zurückgeliefert. Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UGenerateKeyRSA( void* unicon_generator,
                     char* id,
                     char* password,
                     int len  );
```

Die Funktion generiert ein aus Public Key und Secret Key bestehendes RSA–Schlüsselpaar mit der Schlüsselkennung `id` und der Länge `len` Byte. Die beiden Schlüsselkomponenten werden mit dem Paßwort `password` geschützt und in einer Datei gesichert, es sei denn, die gewünschte Kennung ist bereits vergeben. In diesem Fall wird 1 zurückgegeben, sonst 0. Vor der Speicherung erfolgt eine Formatierung der Schlüsselkomponenten mittels `UConvertBinToAscii()`.

```
int UEncryptRSAConverted( void* unicon_generator,
                          char* plaintext,
                          char* id,
                          char* &ciphertext );
```

Die Funktion verschlüsselt den Klartext `plaintext` mit dem Schlüssel der Kennung `id` (Public Key) unter Verwendung des RSA–Chiffrierverfahrens. Vor der Verwendung des Schlüssels wird dieser intern mit `UConvertAsciiToBin()` in sein ursprüngliches Format zurückgeführt. Als Ergebnis wird der mit `UConvertBinToAscii()` nachbearbeitete Schlüsseltext `ciphertext` zurückgeliefert. Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UDecryptRSAConverted( void* unicon_generator,
                          char* ciphertext,
                          char* id,
                          char* &plaintext );
```

Diese Methode stellt die Umkehrfunktion zu `UEncryptRSAConverted()` dar. Das Chiffrat `ciphertext` wird mit dem Schlüssel der Kennung `id` (Secret Key) unter Verwendung des RSA–Chiffrierverfahrens entschlüsselt. Vor der Verwendung des Schlüssels wird dieser intern mit `UConvertAsciiToBin()` in sein ursprüngliches Format zurückgeführt; gleiches gilt für das Chiffrat. Als Ergebnis wird der Klartext `plaintext` zurückgeliefert.Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UEncryptHybridConverted( void* unicon_generator,
                             char* plaintext,
                             char* id,
                             int mode,
                             char* &ciphertext,
                             char* &cipherkey );
```

Die Funktion verschlüsselt den Klartext `plaintext` mit einem zufällig erzeugten temporären Sitzungsschlüssel, wobei das symmetrische Blockverschlüsselungsverfahren IDEA in der Betriebsart `mode` verwendet wird (siehe auch `UGenerateKeyIDEAConverted()`). Das mit `UConvertBinToAscii()` nachbearbeitete Ergebnis wird in `ciphertext` zurückgeliefert. Der temporäre Sitzungsschlüssel wird mittels des RSA–Verschlüsselungsverfahrens unter Verwendung des Schlüssels mit der Kennung `id` (Public Key) asymmetrisch verschlüsselt und dann in `cipherkey` zurückgegeben. Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UDecryptHybridConverted( void* unicon_generator,
                             char* ciphertext,
                             char* cipherkey,
                             char* id,
                             int mode,
                             char* &plaintext );
```

Die Funktion bildet die Umkehrfunktion zu `UEncryptHybridConverted`. Sie entschlüsselt zunächst den Schlüssel `cipherkey` mittels des RSA–Verschlüsselungsverfahrens unter Verwendung des Schlüssels mit der Kennung `id` (Secret Key). Der dabei ermittelte Schlüssel wird für die Dechiffrierung des Schlüsseltextes `ciphertext` mittels des symmetrischen Blockverschlüsselungsverfahrens IDEA in der Betriebsart `mode` verwendet (siehe auch `UGenerateKeyIDEA()`). Zuvor wurde `ciphertext` intern mit `UConvertAsciiToBin()` in sein ursprüngliches Format zurückgeführt. Die Betriebsart muß dieselbe sein, die auch bei der Verschlüsselung eingesetzt wurden. Als Ergebnis wird der Klartext `plaintext` zurückgeliefert. Bei nicht gesetztem Schlüssel wird 1 zurückgegeben, sonst 0.

```
int UGenerateRandom( void* unicon_generator,
                     int len,
                     unsigned char* &result );
```

Die Funktion erzeugt eine Zufallszeichenkette der Länge `len`. Ein Null–Zeichen als Abschluß wird gleich mitgeneriert, damit man einfacher auf char* casten kann. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
void UEncryptMD5( void* unicon_generator,
                  unsigned char* plaintext,
                  int plainlen,
                  unsigned char* &hashtext,
                  int hashlen );
```

Die Funktion verhält sich exakt so wie `UEncryptMD5Converted()`, außer daß auf die Konvertierung zwischen Binär– und ASCII–Format verzichtet wird. Da sowohl `plaintext` als auch `hashtext` vom Typ `unsigned char*` sind, muß jeweils zusätzlich die Länge der Zeichenkette angegeben werden. Analog gibt es folgende Methoden, die jeweils mit der zugehörigen Methode mit Suffix `Converted` korrespondieren (Parameter analog):

```
int UEncryptIDEA();
int UDecryptIDEA();
int UEncryptRSA();
int UDecryptRSA();
int UEncryptHybrid();
int UDecryptHybrid();
```

Diese Funktionen ist flexibler einsetzbar als ihre Pendants, da die implizite Konvertierung entfällt. Außerdem realisieren sie unverfälscht MD5–, IDEA– bzw. RSA–Ergebnisse. Der Nachteil dabei ist, daß 4GL–Sprachen wie z. B. Access Basic nicht zwischen `char*` und `unsigned char*` unterscheiden können. Um verwertbare Ergebnisse zu erzielen, müssen aus solchen Sprachen heraus die Funktionen mit Suffix `Converted` verwendet werden.

Initialisierungsvektoren vs. Abgleich

Betriebsarten wie Cipher Block Chaining (CBC) oder Cipher Feed Back (CFB) von Blockchiffrierverfahren wie IDEA arbeiten i. a. mit sog. Initialisierungsvektoren (IV), um jede verschlüsselte Nachricht einzigartig zu machen. Für jede zu verschlüsselnde Nachricht wird ein zufälliger IV erzeugt (z. B. über Zeitstempel) und als erster Block verschlüsselt. Nachfolgende Blöcke werden je nach IV unterschiedlich verschlüsselt. Zum Dechiffrieren ist der (öffentliche) IV erforderlich. Ohne IV bzw. mit konstantem IV würden identische Nachrichten zu identischen Chiffraten führen. Im Kontext der Kontrollnummergenerierung ist dieser Effekt jedoch zwingend notwendig. Ein Abgleich über chiffrierte Kontrollnummern kann natürlich nicht funktionieren, wenn identische Klartexte auf unterschiedliche Chiffrate abgebildet werden. Aus diesem Grund wird im UNICON–Projekt ein konstanter IV verwendet.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) empfiehlt die Verwendung der Betriebsarten CBC oder CFB. Beide erfüllen in vergleichbarem Maße die gestellten Sicherheits– und Effizienzanforderungen. Bei CBC ist das Chiffrat bis zu einem Block länger als der Klartext, während bei CFB die Längen von Klartext und Chiffrat identisch sind. Bedenkt man, daß mit MD5 und IDEA chiffrierte Kontrollnummern mit UConvertBinToAscii() konvertiert werden und ihre Länge damit um 25% plus (i. a.) 3 Zeichen für die Längenangabe wächst, ergibt sich für CBC eine Gesamtlänge von 33 Zeichen pro Kontrollnummer, bei CFB 23 Zeichen. Bei Verwendung von CFB lassen sich also 30% Speicherplatz einsparen — eine signifikante Reduktion, die Ressourcen spart und den Abgleich beschleunigt. Im UNICON–Projekt wird daher die Betriebsart CFB benutzt.

3.2.3 Die Schnittstelle `CacrymComplexCrypt`

`CacrymComplexCrypt` ist Subklasse von `CacrymSimpleCrypt` und stellt einige kryptographische Funktionen bereit, die sich i. a. aus in `CacrymSimpleCrypt` enthaltenen Verfahren zusammensetzen. Dabei spielt im Kontext des UNICON–Projektes nur die Generierung der Baden–Württembergischen Kontrollnummern eine Rolle.

Basisklasse

`CacrymSimpleCrypt`

Methoden

```
char* UHashBW( void* unicon_generator,
               char* plaintext );
```

Die Funktion erzeugt für Attributwerte in fest vorgegebenem Format eine Kontrollnummer nach dem für das Krebsregister Baden–Württemberg festgeschriebenen Verfahren. Der Eingabewert `plaintext` hat das Format „`VVVNNNTTMMJJJJ` “, wobei „`VVV` “ die ersten drei Buchstaben des Vornamens,

„NNN “ die ersten drei Buchstaben des Nachnamens bzw. Geburtsnamens und „TTMMJJJJ “ das vollständige Geburtsdatum mit vierstelligem Jahr und ggf. führenden Nullen oder Blanks bei Tag und Monat darstellt. Die generierte Kontrollnummer (im Fehlerfall „9999999999999999“) wird zurückgegeben.

3.2.4 Die Schnittstelle UniconBase

UniconBase ist Subklasse von BasisFunktionen, einer Bibliothek von allgemein verwendbaren Funktionen zur Manipulation von Zeichenketten. Im Kontext des UNICON–Projektes spielen daraus vor allem die Methoden zur Standardisierung personenidentifizierender Daten als Voraussetzung zur Generierung von Kontrollnummern eine Rolle. Strenggenommen sind von den nachfolgend beschriebenen Funktionen einige, nämlich UConvertToUpper(), USubstituteSubstring(), URemoveChars() und UExtractSubstring() nicht in UniconBase deklariert, sondern werden von BasisFunktionen geerbt. Es gibt allerdings auch Funktionen in BasisFunktionen, die im UNICON–Projekt nicht benötigt werden und daher an dieser Stelle auch nicht dokumentiert werden.

Basisklasse

BasisFunktionen

Methoden

```
int UAnalyzeName( void* unicon_generator,
                  char* name,
                  int type,
                  int title_possibly_included );
```

Diese Funktion analysiert die Zeichenkette name, zerlegt sie in ihre Komponenten und speichert diese intern, so daß sie anschließend den UniconGenerator–Funktionen UGenerateControlNumber() und UGenerateAllControlNumbers() zur Verfügung stehen. type gibt an, welcher Namenstyp in name enthalten ist. Erlaubte Ausprägungen dabei sind 0 (Name), 1 (Vorname), 2 (Geburtsname), 3 (früherer Name) und 4 (Titel). Falls title_possibly_included ungleich 0 ist, werden Nach–, Vor–, Geburts– und früherer Name daraufhin untersucht, ob sie ggf. eine Angabe des Titels enthalten. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int USetDDRNamecode( void* unicon_generator,
                     char* ddr_namecode );
```

Die Funktion speichert den übergebenen 4–stelligen DDR–Namenscode (z. B. „3238“) intern, so daß er anschließend den UniconGenerator–Funktionen UGenerateControlNumber() und UGenerateAllControlNumbers() zur Verfügung steht. Wird diese Funktion vor einer Kontrollnummergenerierung

nicht aufgerufen, so wird der DDR-Namenscode im Rahmen der Kontrollnummergenerierung implizit aus dem Namen — falls angegeben — berechnet. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int USetDateOfBirth( void* unicon_generator,
                     int day,
                     int month,
                     int year );
```

Die Funktion speichert das übergebene Datum intern, so daß es anschließend den `UniconGenerator`–Funktionen `UGenerateControlNumber()` und `UGenerateAllControlNumbers()` zur Verfügung steht. Die Jahresangabe muß 4–stellig sein. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
void UResetPersonalData( void* unicon_generator );
```

Die Funktion löscht alle intern gespeicherten Personendaten (Namen, Titel, DDR–Namenscode, Geburtsdatum) und sollte aufgerufen werden, nachdem die Kontrollnummern zu einem Patienten erzeugt worden sind und bevor die Personendaten eines anderen Patienten an `UniconBase` übergeben werden.

```
char *URemoveUmlaute( void* unicon_generator,
                      char* string );
```

Die Methode wandelt alle Umlaute aus `string` in „`ae`“, „`oe`“, „`ue`“, „`Ae`“, „`Oe`“ bzw. „`Ue`“ um. „ß“ wird zu „`ss`“.

```
char *UGenerateDDRNamecode( void* unicon_generator,
                            char* surname,
                            char* christian_name );
```

Die Funktion generiert aus den beiden übergebenen Namensteilen `surname` und `christian_name` den vierstelligen DDR–Namenscode (z. B. „`3238` gemäß der Übersetzungstabelle des Gemeinsamen Krebsregisters der neuen Bundesländer und Berlins. Dieser Code wird zurückgeliefert.

```
char *UCologne Phonetics( void* unicon_generator,
                          char* string );
```

Die Funktion erzeugt den Wert der Kölner Phonetik nach dem Verfahren von 1975 für `string` und gibt diesen als Zeichenkette zurück.

```
void USetControlNumber( void* unicon_generator,
                        int index,
                        char* cn );
```

Die Funktion schreibt die Kontrollnummer `cn` unter dem Index `index` (1 – 22) in eine interne Datenstruktur, so daß sie anschließend z. B. zur Generierung eines Kontrollnummeraggregates zur Verfügung steht (siehe `UConvertLinkageToStorage()`).

```
char* UGetControlNumber( void* unicon_generator,
                         int index );
```

Die Funktion liest die Kontrollnummer `cn` mit dem Index `index` (1 – 22) aus. Sie sollte aufgerufen werden, nachdem Kontrollnummern generiert (siehe `UGenerateControlNumber()`) oder aus einem Kontrollnummeraggregat extrahiert worden sind (siehe `UConvertStorageToLinkage()`), um die Kontrollnummern weiterverarbeiten zu können (z. B. Speichern in der Datenbank).

```
char* UConvertToUpper( void* unicon_generator,
                       char* string );
```

Die Funktion konvertiert die Zeichenkette `string` in Großbuchstaben. Umlaute werden korrekt umgesetzt, ß bleibt unverändert. Das Ergebnis wird zurückgeliefert.

```
char *USubstituteSubstring( void* unicon_generator,
                            char* string,
                            char* src,
                            char* dest );
```

Die Funktion verändert die Zeichenkette `string` folgendermaßen: Alle Zeichenfolgen in `string` der Form `src` werden durch Zeichenfolgen der Form `dest` ersetzt. Das Ergebnis wird zurückgeliefert. `USubstituteSubstring()` wird intern von `RemoveUmlaute()` und `UStandardizeBW()` sowie bei der Berechnung phonetischer Codes verwendet.

```
char *URemoveChars( void* unicon_generator,
                    char* string,
                    char* chars );
```

Die Funktion löscht aus der Zeichenkette `string` die in `chars` enthaltenen Zeichen heraus und gibt das Ergebnis zurück. `URemoveChars()` wird intern von `UStandardizeBW()` verwendet.

```
char *UExtractSubstring( void* unicon_generator,
                         char* string,
                         int first,
                         int last );
```

Die Funktion beschränkt die Zeichenkette `string` auf diejenigen Zeichen, die in `string` zwischen den Stellen `first` und `last` stehen (jeweils inklusive). Das erste Zeichen befindet sich dabei an der Stelle 1. Das Ergebnis wird zurückgeliefert. Liegt das Intervall nicht oder nicht vollständig im Bereich der Länge von `string`, so wird das Ergebnis mit Leerzeichen aufgefüllt, so daß er die Länge `(last - first + 1)` hat. Folgendes Beispiel soll das verdeutlichen.

```
input = "schrankwand"

output = UExtractSubstring( input, 1, 5 )   -> output = "schra"
output = UExtractSubstring( input, 8, 13 )  -> output = "wand  "
output = UExtractSubstring( input, 14, 16 ) -> output = "   "
```

3.2.5 Die Schnittstelle `UniconGenerator`

`UniconGenerator` ist Subklasse von `UniconBase` und `CacrymComplexCrypt`. Sie stellt Funktionen zur Generierung und Chiffrierung von Kontrollnummern bereit und bildet die Top–Level–Schnittstelle für das UNICON–Modul. Der Benutzer sollte also vorwiegend diese Klasse verwenden, es sei denn, die angebundene Applikation erlaubt keine Aufrufe von C– bzw. C++–Funktionen. In diesem Fall muß mit der Dateischnittstelle `UniconFile` (s. u.) Vorlieb genommen werden. Benutzern von `UniconGenerator` stehen durch das Vererbungsprinzip zusätzlich zu den in dieser Klasse deklarierten Funktionen alle öffentlichen Funktionen der Basisklassen zur Verfügung, also sowohl die Kryptographie– als auch die Standardisierungsfunktionen.

Basisklassen

`UniconBase`
`CacrymComplexCrypt`

Methoden

```
void* UInitGenerator( char *id,
                      char *password );
```

Diese Funktion ist vor der Nutzung der Funktionsschnittstelle des UNICON–Moduls einmalig aufzurufen. Ihr sind Identifikation und Paßwort des Benutzers bzw. des zu verwendenden (IDEA–)Schlüssels zu übergeben. Wird bei der Authorisierungsüberprüfung festgestellt, daß es sich nicht um einen authorisierten Benutzer bzw. Schlüssel handelt, wird intern ein Fehlerstatus auf 1 gesetzt, womit die Benutzung der Funktionen dieser Klasse unterbunden wird. Ein Fehlerstatus von 0 bedeutet, daß die Authorisierung erfolgreich war. Der Fehlerstatus kann mit der Methode `UGetErrorState()` abgefragt werden. Optional kann auf Identifikation und Paßwort auch verzichtet werden, indem beide Parameter auf `NULL` gesetzt werden. In diesem Fall werden Kontrollnummern nur mit MD5 chiffriert, also mit einem Einwegverfahren, das keinen Schlüssel erfordert. Auf die IDEA–Überverschlüsselung wird hingegen verzichtet. Diese Option ist für Baden–Württemberg relevant, da dort eine dezentrale Verschlüsselung von Kontrollnummern erforderlich ist (MD5 beim Melder, IDEA beim Statistischen Landesamt). `UInitGenerator()` liefert ein Handle zurück, das allen anderen Funktionen des UNICON–Moduls als erster Parameter übergeben werden muß. Das Handle ist so lange gültig, bis `UFinishGenerator()` aufgerufen wird.

```
void UFinishGenerator( void* unicon_generator );
```

Die Funktion schließt die mit `UInitGenerator()` angelegte UNICON–Umgebung. Das übergebene Handle ist anschließend nicht mehr gültig.

```
int UGetErrorState( void* unicon_generator );
```

Die Funktion gibt den aktuellen Wert des bei der Initialisierung gesetzten Fehlerstatus zurück. Dabei bedeutet 0, daß die Authorisierung des Benutzers erfolgreich war, und 1, daß die Authorisierung fehlgeschlagen ist.

```
char *UEncryptStandardizedInput( void* unicon_generator,
                                 char* plaintext );
```

Der noch unverschlüsselte Wert der Kontrollnummer `plaintext` wird zunächst mit dem Hashverfahren MD5 und anschließend mit dem symmetrischen Chiffrierverfahren IDEA verschlüsselt. `UEncryptStandardizedInput()` ist für die Kontrollnummern 1 bis 20 geeignet, nicht jedoch für die baden–württembergischen Kontrollnummern 21 und 22. Für letztere gibt es die spezielle Methode `UEncryptStandardizedInputBW()` (s. u.). Das Ergebnis der Verschlüsselung wird zurückgegeben.

```
char *UEncryptStandardizedInputBW( void* unicon_generator,
                                   char* plaintext );
```

Die Funktion ruft erzeugt zunächst nach dem für das Krebsregister Baden–Württemberg festgeschriebenen Verfahren aus `plaintext` eine chiffrierte Kontrollnummer. `plaintext` sollte dazu das Format „`VVVNNNTTMMJJJJ` “ haben, wobei „`VVV` “ die ersten drei Buchstaben des Vornamens, „`NNN` “ die ersten drei Buchstaben des Nachnamens bzw. Geburtsnamens und „`TTMMJJJJ` “ das vollständige Geburtsdatum mit vierstelligem Jahr und ggf. führenden Nullen bei Tag und Monat darstellt. Tritt bei der Chiffrierung ein Fehler auf, so liefert `UEncryptStandardizedInputBW()` als Ergebnis die chiffrierte „leere“ Kontrollnummer zurück, die auch mit der Methode `UGetEmptyControlNumber()` abgefragt werden kann. Ansonsten wird die chiffrierte Kontrollnummer mittels IDEA überverschlüsselt und das Resultat zurückgegeben. `UEncryptStandardizedInputBW()` ist nur für die Kontrollnummern 21 und 22 geeignet, nicht jedoch für die Kontrollnummern 1 bis 20. Für letztere gibt es die Methode `UEncryptStandardizedInput()` (s. o.).

```
int UGenerateControlNumber( void* unicon_generator,
                            int index,
                            int no_encryption );
```

Die Funktion generiert anhand der intern in `UniconBase` gespeicherten personenidentifizierenden Informationen zu einem Patienten diejenige Kontrollnummer, die durch `index` festgelegt ist. `index` muß zwischen 1 und 22 liegen (jeweils inklusive). Intern werden die für die jeweilige Kontrollnummer benötigten Angaben ausgelesen und mittels `RemoveUmlaute()` bzw. `UStandardizeBW()` aus `UniconBase` standardisiert. Anschließend erfolgt eine Chiffrierung mittels `UEncryptStandardizedInput()` bzw. `UEncryptStandardizedInputBW()`, es sei denn, `no_encryption` hat den Wert 1. Das Ergebnis wird intern gespeichert und kann später mittels `UGetControlNumber()` ausgelesen werden. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int UGenerateAllControlNumbers( void* unicon_generator,
                                int no_encryption );
```

Die Funktion generiert anhand der intern in `UniconBase` gespeicherten personenidentifizierenden Informationen zu einem Patienten alle 22 Kontrollnummern. Intern werden die für die jeweilige Kontrollnummer benötigten Angaben ausgelesen und mittels `RemoveUmlaute()` bzw. `UStandardizeBW()` aus `UniconBase` standardisiert. Anschließend erfolgt eine Chiffrierung mittels `UEncryptStandardizedInput()` bzw. `UEncryptStandardizedInputBW()`, es sei denn, `no_encryption` hat den Wert 1. Das Ergebnis wird intern gespeichert und kann später mittels `UGetControlNumber()` ausgelesen werden. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
char *UGetEmptyControlNumber();
```

Die Funktion liefert dasjenige Chiffrat zurück, welches mit der leeren Kontrollnummer („“) assoziiert ist.

```
int UConvertControlNumber( void* unicon_generator,
                           char* old_cn,
                           char* new_id,
                           char* &new_cn );
```

Die Funktion konvertiert die Kontrollnummer `old_cn`, die mit dem bei `UInitGenerator()` als Parameter angegebenen IDEA–Schlüssel chiffriert sein muß. Dabei wird die bisherige IDEA–Verschlüsselung rückgängig gemacht und eine neue IDEA–Verschlüsselung mit dem zu `new_id` gehörigen Schlüssel durchgeführt, der zuvor mit `SetKey()` zu setzen und anschließend mit `UnsetKey()` wieder freizugeben ist. Die konvertierte Kontrollnummer wird in den Referenzparameter `new_cn` eingetragen. Im Fehlerfall wird 1 zurückgegeben, sonst 0. Die Funktion eignet sich zur Vorbereitung eines Abgleichs zwischen verschiedenen Krebsregistern bzw. für einen Bundesabgleich.

```
int UConvertLinkageToStorage( void* unicon_generator,
                              char* id,
                              char* empty_cn,
                              char* &storage );
```

Die Funktion wandelt Kontrollnummern, die intern in der UNICON–Umgebung gespeichert sein müssen (siehe `GenerateControlNumber()` oder `SetControlNumber()`), vom Linkage– in das Storage–Format um. Das Ergebnis wird in `storage` eingetragen. Die IDEA–Überverschlüsselung wird mit dem zu `id` gehörigen Schlüssel durchgeführt, der zuvor mit `SetKey()` zu setzen und anschließend mit `UnsetKey()` wieder freizugeben ist. Als weitere Information muß die mit dem Leerstring (Länge 0) assoziierte Kontrollnummer `empty_cn` übergeben werden. Im Fehlerfall wird 1 zurückgegeben, sonst 0. Gemäß den in [AMST96] ausgesprochenen Empfehlungen sollte die Registerstelle Kontrollnummern dauerhaft nur im Storage–Format speichern.

```
int UConvertStorageToLinkage( void* unicon_generator,
                              char* id,
                              char* empty_cn,
                              char* storage );
```

Die Funktion wandelt Kontrollnummern vom Storage– in das Linkage–Format um, i. a. als Vorbereitung eines bevorstehenden Abgleichs. Die einzelnen Kontrollnummern werden aus dem Aggregat `storage` extrahiert und intern in der UNICON–Umgebung gespeichert. Sie können anschließend mit `GetControlNumber()` ausgelesen werden. Die IDEA–Entschlüsselung wird mit dem zu `id` gehörigen Schlüssel durchgeführt, der zuvor mit `SetKey()` zu setzen und anschließend mit `UnsetKey()` wieder freizugeben ist. Als weitere Information muß die mit dem Leerstring (Länge 0) assoziierte Kontrollnummer `empty_cn` übergeben werden. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int UEncryptPersonalData( void* unicon_generator,
                          char *plaintext,
                          char *rsa_id,
                          char *&ciphertext );
```

Die Funktion nimmt eine hybride Chiffrierung von `plaintext` vor. Im Krebsregisterkontext handelt es sich bei `plaintext` i. a. um eine Verkettung von personenbezogenen Daten, zur späteren Zerlegung häufig durch ein Trennzeichen separiert (z. B. `Hans#Meier#Dorfstr. 12#12345 Kleckersdorf`). Die Verkettung ist *vor* dem Funktionsaufruf vom Benutzer durchzuführen. Prinzipiell können der Funktion beliebige Zeichenketten übergeben werden. Die Chiffrierung erfolgt anhand des zu `rsa_id` gehörigen RSA–Schlüssels (public key), der zuvor mit `SetKey()` zu setzen und anschließend mit `UnsetKey()` wieder freizugeben ist. Das Ergebnis wird in `ciphertext` gespeichert. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int UDecryptPersonalData( void* unicon_generator,
                          char *ciphertext,
                          char *rsa_id,
                          char *&plaintext );
```

Die Funktion nimmt eine hybride Dechiffrierung von `ciphertext` vor. Die Dechiffrierung erfolgt anhand des zu `rsa_id` gehörigen RSA–Schlüssels (secret key), der zuvor mit `SetKey()` zu setzen und anschließend mit `UnsetKey()` wieder freizugeben ist. `ciphertext` muß ursprünglich als Ergebnis eines Aufrufs von `UEncryptPersonalData()` hervorgegangen sein. Das Ergebnis wird in `plaintext` gespeichert. Im Krebsregisterkontext handelt es sich bei `plaintext` i. a. um eine Verkettung von personenbezogenen Daten (s. o.), die nach erfolgter Dechiffrierung nun wieder vom Benutzer in einzelne Bestandteile zerlegt werden kann. Im Fehlerfall wird 1 zurückgegeben, sonst 0.

```
int UEncryptControlNumberBW( void* unicon_generator,
                             char *bw_cn,
                             char *&ciphertext );
```

Die Funktion führt eine IDEA–Überverschlüsselung auf einer Baden–Württemberger Kontrollnummer aus, die als 16–stellige Ziffernfolge in `bw_cn` zu übergeben ist. Als Schlüssel wird derjenige verwendet, dessen ID und Paßwort der Funktion `UInitGenerator()` übergeben wurde. Das Ergebnis wird in `ciphertext` gespeichert. Im Fehlerfall wird 1 zurückgegeben, sonst 0. Die Funktion wird ausschließlich vom Krebsregister Baden–Württemberg benötigt, um die dort verwendeten Kontrollnummern für einen länderübergreifenden Abgleich vorzubereiten.

```
int UEncryptControlNumberMD5( void* unicon_generator,
                              char *md5_cn,
                              char *&idea_cn );
```

Die Funktion führt eine IDEA–Überverschlüsselung auf der Kontrollnummer `md5_cn` aus, die nur mit MD5 verschluesselt sein darf. Als Schlüssel wird derjenige verwendet, dessen ID und Paßwort der Funktion `UInitGenerator()` übergeben wurde. Das Ergebnis wird in `idea_cn` gespeichert. Im Fehlerfall wird 1 zurückgegeben, sonst 0. Die Funktion wird insbesondere vom Krebsregister Baden–Württemberg benötigt, da dort Kontrollnummern dezentral zu verschluesseln sind (MD5 beim Melder, IDEA beim Statistischen Landesamt). Aber auch zur Konvertierung von Kontrollnummern, die mit einer zu UNICON inkompatiblen Krypto–Bibliothek IDEA–chiffriert wurden, kann die Funktion hinzugezogen werden.

3.2.6 Beispiel

Beispielhaft läuft die Generierung von Kontrollnummern mit `UniconGenerator` wie folgt ab:

```
//----- Umgebung anlegen... -----
int handle = UInitGenerator( my_id, my_password );

//----- Pruefen, ob Authorisierung geklappt hat... -----
int error = UGetErrorState( handle );
if( error )
  <abbrechen>;

//----- Leere KN merken -----
char* leere_kn = UGetEmptyControlNumber( handle );

<ersten Patienten p waehlen>;

while( <p vorhanden> )
{
  //----- ACHTUNG: Vernachlaessigte Fehlerbehandlung! -----
  if( <p.nachname vorhanden> )
    error = UAnalyzeName( handle, p.nachname, 0, 0 );
```

```
  //----- Im Vornamen soll nach Titeln gesucht werden... -----
  if( <p.vorname vorhanden> )
    error = UAnalyzeName( handle, p.vorname, 1, 1 );

  if( <p.geburtsname vorhanden> )
    error = UAnalyzeName( handle, p.geburtsname, 2, 0 );

  if( <p.frueherer_name vorhanden> )
    error = UAnalyzeName( handle, p.frueherer_name, 3, 0 );

  if( <p.geburtsdatum vorhanden> )
    error = USetDateOfBirth( handle, p.geburtsdatum.tag,
             p.geburtsdatum.monat, p.geburtsdatum.jahr );

  //----- DDRNamenscode implizit erzeugen -----

  // ----- Dann die Kontrollnummern generieren... -----
  // ----- ...und abspeichern (ggf. Leerstring)   -----
  for( int i = 1; i <= 22; i++ )
  {
    error = UGenerateControlNumber( handle, i, 0 );

    char *kn = UGetControlNumber( handle, i );
    if( <kn gleich leere_kn> )
      <speichere> "";
    else
      <speichere> kn;
  } //for

  // ----- Jetzt den internen Speicher loeschen... -----
  UResetPersonalData( handle );

  <naechsten Patienten p waehlen>;
} //while

//----- Umgebung wieder freigeben... -----
UFinishGenerator( handle );
```

3.3 Die Dateischnittstelle

Neben der Funktionsschnittstelle stellt UNICON eine Dateischnittstelle zur Generierung von Kontrollnummern bereit. Diese ist von denjenigen Applikationen zu verwenden, die keine Aufrufe von C– oder C++–Funktionen erlauben und daher die UNICON–Funktionsschnittstelle nicht benutzen können. Im folgenden soll beispielhaft die Verwendung der Dateischnittstelle beschrieben werden. Dieser Abschnitt ist vor allem für den Endbenutzer von Bedeutung.

Nachdem die personenidentifizierenden Daten eines Meldebogens am Rechner eingegeben worden sind, soll das Erfassungsprogramm die Erzeugung der Kontrollnummern zu diesem Datensatz veranlassen. Dazu werden die zur Generierung der Kontrollnummern benötigten Attributwerte in eine ASCII–Datei (Eingabedatei) geschrieben. Anschließend wird aus der Applikation heraus per System Call ein ausführbares Programm aufgerufen, welches den Dateiinhalt liest, die Kontrollnummern bildet und diese in eine zweite ASCII–Datei (Ausgabedatei) einträgt. Letztere wird von der Applikation eingelesen, welche die Kontrollnummern daraufhin weiterverarbeiten kann. Die Dateinamen für Ein– und Ausgabedatei sind vom Benutzer frei wählbar und müssen als Parameter an das aufgerufene Programm übergeben werden.

Sollen für in der Vergangenheit bearbeitete Datensätze nachträglich Kontrollnummern generiert werden, so wäre dafür eine Einschränkung, daß die Eingabedatei nur einen einzigen Datensatz beinhalten darf, hinderlich. Daher kann die Eingabedatei beliebig viele Datensätze aufnehmen. Das Programm zur Kontrollnummergenerierung erkennt die Anzahl der Datensätze automatisch, erzeugt entsprechend viele Datensätze mit Kontrollnummern und trägt diese in die Ausgabedatei ein. Die Applikation muß sich merken, wieviele Datensätze in die Eingabedatei eingetragen wurden, um später die korrekte Anzahl von Kontrollnummernsätzen auslesen zu können. Der i–te Kontrollnummernsatz in der Eingabedatei bezieht sich jeweils auf den i–ten Datensatz in der Ausgabedatei.

Die Dateischnittstelle beinhaltet neben dem ausführbaren Programm zur Generierung von Kontrollnummern ein weiteres Programm zur Schlüsselverwaltung. Zur Chiffrierung der Kontrollnummern mit dem IDEA–Verfahren muß z. B. ein entsprechender IDEA–Schlüssel zur Verfügung stehen. Dieser ist vor dem erstmaligen Einsatz des UNICON–Moduls einmal zu erzeugen. Alle Kontrollnummern müssen natürlich mit demselben Schlüssel generiert werden, damit sie miteinander abgeglichen werden können. Der Schlüssel wird in einer ASCII–Datei abgelegt und bei Bedarf vom Programm zur Kontrollnummergenerierung ausgelesen. Aus Sicherheitsgründen darf auf den Schlüssel nur über eine Schlüssel–ID und ein Paßwort zugegriffen werden. ID und Paßwort werden vom Benutzer festgelegt und an das Programm zur Schlüsselerzeugung übergeben. Das Paßwort kann und sollte regelmäßig geändert werden. Auch das Programm zur Kontrollnummergenerierung muß als Parameter ID und Paßwort des zu verwendenden Schlüssels übergeben bekommen.

3.3.1 UNICON File Interface

Im folgenden werden die Aufrufparameter des UNICON File Interface (Dateiname unter MS–DOS und Windows 3.1: UNICONFI.EXE, sonst UniconFileInterface) erläutert. Zusätzlich werden jeweils die Dateiformate für Ein– und Ausgabe angegeben. Dabei wird jeweils nur ein Datensatz beispielhaft aufgeführt. Beliebig viele Datensätze können in unmittelbarem Anschluss folgen (nächste Zeile, angedeutet durch „...“).

Kontrollnummergenerierung mit IDEA–Überverschlüsselung

```
UNICONFI g <input_file> <output_file> <id> <password>
```

Klartext–Datensätze werden aus `<input_file>` ausgelesen. Aus diesen werden Kontrollnummern erzeugt, wobei zur Überverschlüsselung ein IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` verwendet wird. Die Kontrollnummern werden in `<output_file>` geschrieben.

Eingabeformat (F1)

```
<ID>
<Name>
<Vorname>
<Geburtsname>
<Frueherer Name>
<Vollstaendiges Geburtsdatum (TTMMJJJJ)>
<DDR-Namenscode>
<Titel>
...
```

Es müssen nicht immer alle Zeilen eines Datensatzes ausgefüllt sein. Fehlt z. B. der Geburtsname, kann die entsprechende Zeile leer bleiben (nur Line Feed unter UNIX, sonst zusätzlich Carriage Return). Die Namensangaben sowie der Titel können aus mehreren Komponenten bestehen, die durch die Trennzeichen Blank (32), Bindestrich (45), Punkt (46), Doppelpunkt (58), Komma (44), Semikolon (59) oder Apostroph (39) zu separieren sind (in Klammern die dezimalen ASCII–Codes). Das Geburtsdatum ist im Format `TTMMJJJJ` anzugeben, also ggf. mit führenden Nullen bei Tag und Monat sowie mit vierstelligem Jahr. Der DDR–Namenscode muß vierstellig sein. Wird er nicht angegeben, wird er intern automatisch aus dem Namen und Vornamen erzeugt. Titel können außer in der dedizierten Zeile auch in der Zeile für Vornamen[2] stehen. Wichtig ist, daß Titelangaben pro Datensatz nur in maximal *einer* Zeile vorkommen dürfen, also nur entweder in der Vornamens– oder der Titelzeile. In den Zeilen für Namen, Vornamen, Geburtsnamen und frühere Namen wird automatisch nach Füllseln gesucht. Die Angabe der ID (eines Patienten) ist ebenfalls optional; sie wird — falls angegeben — unverändert in die Ausgabedatei übernommen und dient lediglich der Erleichterung der Orientierung des Benutzers.

Jeweils acht Zeilen der Eingabedatei bilden einen Datensatz. Weitere Datensätze schließen sich direkt daran an. So beginnt z. B. der zweite Datensatz in der neunten Zeile, der dritte in der 17. usw. Auch die letzte Zeile des letzten Datensatzes der Datei muß mit einem Line Feed — ggf. mit zugehörigem Carriage Return — abgeschlossen werden. Jede Zeile (mit Ausnahme der Zeile für die ID) darf führende Blanks besitzen; diese werden ignoriert. Die Anzahl der

[2] Bei Verwendung der Funktionsschnittstelle kann das System so konfiguriert werden, daß auch Name, Geburtsname oder früherer Name auf eventuell enthaltene Titelangaben untersucht werden (registerabhängig). Für die Dateischnittstelle ist eine Hardcodierung unumgänglich.

Datensätze einer Eingabedatei ist lediglich durch die zur Verfügung stehende Plattenkapazität beschränkt.

Groß– und Kleinschreibung ist beliebig; intern erfolgt eine Umwandlung in Großbuchstaben. Die Benutzung von Umlauten und „ß“ ist erlaubt. Es werden sowohl Umlaute des unter Windows und UNIX verwendeten ISO 8859–Zeichensatzes als auch des unter MS–DOS und OS/2 gebräuchlichen IBM–Zeichensatzes (Codepages 437 oder 850) erkannt und intern in die zweibuchstabige Schreibweise umgesetzt.

Es folgt ein Beispiel für eine Eingabedatei mit zwei Datensätzen. Die Zeilennummern dienen nur der Orientierung, sie dürfen *auf keinen Fall* in der Eingabedatei enthalten sein.

```
 1: 1234
 2: von Hohen Tiefenstein
 3: Karl Eduard
 4:
 5:
 6: 11051924
 7: 3338
 8: Freiherr
 9: 1235
10: Schroeder von Wegen
11: Eva-Maria Baerbel Brunhild, Prof. Dr.
12: Guenther-Engelbrecht zum Bueschenfelde
13: von Trontheim zu Laubus-Eschbach
14: 07021931
15:
16:
```

Der erste Datensatz (ID 1234) enthält einen Namen mit drei Komponenten, von denen eine (`von`) ein Füllsel darstellt. Es liegen zwei Vornamen vor, jedoch weder Geburtsnamen noch frühere Namen. Der DDR–Namenscode ist explizit angegeben (`3338`); der Titel steht in der dedizierten Zeile.

Im zweiten Datensatz (ID 1235) besteht der Name ebenfalls aus drei Teilen, darunter ein Füllsel. In der Vornamenszeile sind vier Teile des Vornamens sowie zwei Titelteile angegeben. Geburtsname und früherer Name bestehen aus je drei Teilen plus ein bzw. zwei Füllsel. Der DDR–Namenscode fehlt und wird daher automatisch berechnet, bevor die entsprechende Kontrollnummer generiert wird. Da die Vornamenszeile Titel enthält, *muß* die Titelzeile frei bleiben.

Ausgabeformat (F2)

```
<ID>
<Name, Teil 1>
<Name, Teil 2>
<Name, Teil 3>
<Vorname, Teil 1>
```

```
<Vorname, Teil 2>
<Vorname, Teil 3>
<Geburtsname, Teil 1>
<Geburtsname, Teil 2>
<Geburtsname, Teil 3>
<Frueherer Name, Teil 1>
<Frueherer Name, Teil 2>
<Frueherer Name, Teil 3>
<Geburtstag>
<DDR-Namenscode>
<Phonetischer Code des standardisierten Namens>
<Phonetischer Code des standardisierten Vornamens>
<Phonetischer Code des standardisierten Geburtsnamens>
<Phonetischer Code des standardisierten frueheren Namens>
<Titel, Teil 1>
<Titel, Teil 2>
<Baden--Wuerttemberger Kontrollnummer 1>
<Baden--Wuerttemberger Kontrollnummer 2>
...
```

Eine Ausgabedatei hat `23 / 8` mal so viele Zeilen wie die zugehörige Eingabedatei. Jeder Datensatz der Ausgabedatei besteht also aus 23 Zeilen, wobei die erste Zeile optional eine ID (unverändert übernommen aus der Eingabedatei) und jede weitere Zeile eine (ggf. leere) Kontrollnummer enthält.

Diese Struktur entspricht der in den "Empfehlungen an die Bundesländer..." beschriebenen Spezifikation. Zu beachten ist die Reihenfolge der beiden baden–württemberger Kontrollnummern: *als erstes* wird die Kontrollnummer genannt, in die der Geburtsname eingeht, *als zweites* die Kontrollnummer, die vom Namen beeinflußt wird. Dies entspricht der in Baden–Württemberg verwendeten Reihenfolge.

Analog zur Eingabedatei müssen nicht immer alle Zeilen eines Datensatzes Informationen enthalten. Fehlt z. B. der Geburtsname in der Eingabedatei, so bleiben die entsprechenden Zeilen in der Ausgabedatei leer (nur Line Feed unter UNIX, sonst zusätzlich Carriage Return). Auch wenn Angaben, die zur Bildung einer Kontrollnummer erforderlich sind, nur unvollständig vorhanden sind, wird intern eine „leere" Kontrollnummer erzeugt, die bei der Ausgabe in eine Leerzeile umgesetzt wird. Dies ist insbesondere bei aggregierten Kontrollnummern wie dem DDR–Namenscode und den Baden–Württemberger Kontrollnummern zu beachten. Bei der Generierung der Kontrollnummer für den DDR–Namenscode wird wie folgt vorgegangen: Falls angegeben, wird der DDR–Code aus Zeile 7 fuer die Kontrollnummergenerierung verwendet. Ansonsten wird der Code zur Laufzeit aus dem Namen und Vornamen erzeugt. Liegen Name oder Vorname oder beide nicht vor (Leerzeile(n)), wird eine leere Kontrollnummer erzeugt. Liefert der Algorithmus zur Berechnung der Baden–Württemberger Kontrollnummer den Fehlerwert „9999999999999999" zurück, wird ebenfalls eine leere Kontrollnummer gebildet. Beim Geburtsdatum ist unbedingt auf die Angabe einer 4–stelligen Jahreszahl zu achten.

Typischerweise besteht eine Ausgabedatei ca. zur Hälfte aus Leerzeilen, da viele Personen keine zusammengesetzten Namen oder Titel haben bzw. diese nicht immer erfaßt worden sind. Fehlen Geburtsname und / oder früherer Name, so können natürlich auch die zugehörigen phonetischen Codes nicht gebildet werden.

Alle Zeilen der Ausgabedatei (mit Ausnahme der ID–Zeilen) besitzen die gleiche Länge. Diese beträgt bei Verzicht auf IDEA bzw. bei Verwendung von IDEA in der Betriebsart Cipher Feed Back (CFB) 23 Zeichen, bei Verwendung von IDEA in der Betriebsart Cipher Block Chaining (CBC) 33 Zeichen. Aufgrund des geringeren Speicherplatzbedarfs werden Kontrollnummern in UNICON mittels CFB generiert. Ursprünglich sind die von den kryptographischen Algorithmen erzeugten Chiffrate nur 16 (nur MD5 oder zusätzlich IDEA–CFB) bzw. 24 (IDEA–CBC) Zeichen lang, durch die anschließende Konvertierung auf druckbare Zeichen (siehe `UConvertBinToAscii()` aus `CacrymBase`) vergrößern sich die Längen jedoch auf die angegebenen Werte.

Im Gegensatz zu den Kontrollnummern 1 bis 20 sind die Kontrollnummern 21 und 22 (Baden–Württemberg) nicht mit IDEA, sondern „nur“ mit IDEA verschlüsselt, da diese ja selbst schon auf einem MD5–ähnlichen Algorithmus beruhen. Ist bei der Verarbeitung eines Datensatzes ein Fehler aufgetreten (z. B. durch Vorkommen unzulässiger Zeichen), so sind alle 22 zugehörigen Zeilen der Outputdatei leer. Eine Fehlermeldung mit Datensatznummer (laufende Nummer, nicht ID!) wird auf der Konsole ausgegeben.

Im folgenden ist die Ausgabedatei angegeben, die aus der im vorigen Abschnitt vorgestellten beispielhaften Eingabedatei mit zwei Datensätzen erzeugt wurde. Wiederum dienen die Zeilennummern lediglich der Orientierung, sie werden nicht in die Ausgabedatei geschrieben.

```
 1: 1234
 2: \_j34XtNj.!k4-0285!qx16
 3: E]b9[$81h53Up8U>@XK[x16
 4: A&rHkNU'6b;bh'-MJ<=#x16
 5: ;c[t@C:0p9<JldI_MP:jx16
 6: 6.!]L?'CKq.-BsXP'ctmx16
 7:
 8:
 9:
10:
11:
12:
13:
14: ?7(t,1<\P8*#=AR&HndNx16
15: (=V=kW+\W.‘mJ2$a?G.@x16
16: "lfp<KqqLJ5Ts*&='s-Gx16
17: @3;C;+@VYVQ=SIOORPq0x16
18:
19:
20: peaDhYE4oj‘CduF;-P%Kx16
```

```
21:
22:
23: 1235
24: $N_/!gbq_Yg5!&)dC(2Sx16
25: Ug_G*J*1Z2/JKSm;)n0Ix16
26: 4e7kQG--o?$Ku)8)teTIx16
27: A&rHkNU'6b;bh'-MJ<=#x16
28: E0m<g"/6AVefU$3\Q%DTx16
29: g7s^0R'Tr!lXb8sq4$8Dx16
30: 12kKiT_6gUmY88a5)‘!Ix16
31: _b&[9hr?i:eRiHTRujfgx16
32: bj;9n;P"8u^UQ0=mS)u"x16
33: 3_2Qq\Q7%Yb:68/\?'gXx16
34: Y:%dgA%<(ARQ3/?p%eu!x16
35: /;:*Js62>dcjgis-?d0Rx16
36: GhjT1Zll39FHY@_QI/06x16
37: lkqU++!51an27!RpU1s)x16
38: 3YRun\WS3s/q6^.-a'"dx16
39: &kQ;ZN)''1MhC]/U'YI!x16
40: s&Eqlps_+c]//I<D6.W8x16
41: pYTX:M7n@VE8<sp6^&jgx16
42: l)JoQ0lQPJYVZre7\RtHx16
43: d%‘#n4\dPnSj8[3>6+![x16
44: $t''JR‘1/]RROjTpnroXx16
45: #R2A.hD\4fT0hh[7He/.x16
46: #QPbugG;MXa-HG%("e_Cx16
```

Man erkennt, daß der erste Datensatz drei standardisierte Namenskomponenten enthält, während „nur“ zwei Vornamenskomponenten gefunden wurden (Zeile 7 ist leer). Weder Geburtsnamen noch frühere Namen sind vorhanden (Zeilen 8 bis 13). Dementsprechend wurden für diese Attribute keine phonetischen Codes erzeugt (Zeilen 18 und 19). Des weiteren bleibt die Zeile für die zweite Titelkomponente frei. Das gleiche gilt für die erste baden–württemberger Kontrollnummer, da der Geburtsname fehlt.

Der zweite Datensatz (Zeilen 24 bis 46) ist aufgrund seiner Komplexität unrealistisch; alle Kontrollnummern konnten gebildet werden. Auffällig ist, daß die Zeilen 4 und 27 übereinstimmen, da das Füllsel `von` bei beiden Datensätzen die dritte Komponente des standardisierten Namens darstellt.

Kann eine Kontrollnummer nicht gebildet werden, da in der Eingabedatei ein unzulässiges Zeichen vorliegt, z. B. ein Slash (/), so werden für den gesamten zugehörigen Datensatz keine Kontrollnummern in die Ausgabedatei geschrieben. Stattdessen werden nach der ID 22 Leerzeilen eingetragen. Anhand dieses leeren Ausgabedatensatzes kann die aufrufende Applikation erkennen, bei welchen Datensätzen Fehler aufgetreten sind. Nachdem diese beseitigt worden sind (i. a. durch Entfernung oder Ersetzung unzulässiger Zeichen), kann die Kontrollnummergenerierung durchgeführt werden.

Kontrollnummergenerierung ohne IDEA–Überverschlüsselung

```
UNICONFI m <input_file> <output_file>
```

Klartext–Datensätze werden aus `<input_file>` ausgelesen. Aus diesen werden Kontrollnummern erzeugt, wobei ausschließlich mit MD5 verschlüsselt wird. Die Kontrollnummern werden in `<output_file>` geschrieben. Diese Aufrufoption wird insbesondere vom Krebsregister Baden–Württemberg benötigt, da dort Kontrollnummern dezentral zu verschluesseln sind (MD5 beim Melder, IDEA beim Statistischen Landesamt).

Eingabeformat: siehe F1

Ausgabeformat: siehe F2

Kontrollnummergenerierung ohne Verschlüsselung

```
UNICONFI n <input_file> <output_file>
```

Klartext–Datensätze werden aus `<input_file>` ausgelesen. Aus diesen werden Kontrollnummern erzeugt, wobei keine Verschlüsselung vorgenommen wird. Die Kontrollnummern werden in `<output_file>` geschrieben.

Eingabeformat: siehe F1

Ausgabeformat: siehe F2

Aggregierung vom Linkage– ins Storage–Format

```
UNICONFI a <input_file> <output_file> <id> <password>
           <empty_control_number>
```

Kontrollnummern werden aus `<input_file>` ausgelesen. Aus diesen wird pro Datensatz ein Aggregat erzeugt, wobei zur Überverschlüsselung ein IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` verwendet wird. Zusätzlich ist in `<empty_control_number>` diejenige Kontrollnummer zu übergeben, die mit der leeren Zeichenkette assoziiert ist (bezogen auf den verwendeten IDEA–Schlüssel). Die Aggregate werden in `<output_file>` geschrieben.

Eingabeformat: siehe F2

Ausgabeformat (F3)

```
<ID>
<Aggregat>
...
```

Deaggregierung vom Storage– ins Linkage–Format

```
UNICONFI d <input_file> <output_file> <id> <password>
           <empty_control_number>
```

Umkehrfunktion von `UNICONFI a ...`: Aggregate werden aus `<input_file>` ausgelesen. In `<empty_control_number>` ist diejenige Kontrollnummer zu übergeben, die mit der leeren Zeichenkette assoziiert ist (bezogen auf den verwendeten IDEA–Schlüssel). Durch Dechiffrierung erhält man jeweils 22 Kontrollnummern, wobei ein IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` verwendet wird (muß derselbe Schlüssel sein, der zum Aggregieren benutzt wurde). Die Kontrollnummern werden in `<output_file>` geschrieben.

Eingabeformat: siehe F3

Ausgabeformat: siehe F2

Konvertierung von Landes– in Bundesverschlüsselung

```
UNICONFI c <input_file> <output_file> <id> <password>
           <global_id> <global_password>
```

Kontrollnummern werden aus `<input_file>` ausgelesen. Die IDEA–Überverschlüsselung wird jeweils rückgängig gemacht, wobei ein (landesweit einheitlicher) IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` verwendet wird. Anschließend erfolgt eine erneute IDEA–Überverschlüsselung mit einem (bundesweit einheitlichen) IDEA–Schlüssel mit Kennung `<global_id>` und Paßwort `<global_password>`. Die neu verschlüsselten Kontrollnummern werden in `<output_file>` geschrieben.

Eingabeformat: siehe F2

Ausgabeformat: siehe F2

IDEA–Überverschlüsselung von nur mit MD5 chiffrierten Kontrollnummern

```
UNICONFI i <input_file> <output_file> <id> <password>
```

Kontrollnummern, die nur mit MD5 verschlüsselt sein dürfen, werden aus `<input_file>` ausgelesen und mit einem IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` überchiffriert. Die überverschlüsselten Kontrollnummern werden in `<output_file>` geschrieben. Diese Aufrufoption wird insbesondere vom Krebsregister Baden–Württemberg benötigt, da dort Kontrollnummern dezentral zu verschluesseln sind (MD5 beim Melder, IDEA beim Statistischen Landesamt). Aber auch zur Konvertierung von Kontrollnummern, die mit einer zu UNICON inkompatiblen Krypto–Bibliothek IDEA–chiffriert wurden, kann diese Aufrufoption hinzugezogen werden.

Eingabeformat: siehe F2

Ausgabeformat: siehe F2

Hybride asymmetrische Verschlüsselung

```
UNICONFI h <input_file> <output_file> <id> <password>
```

Personenbezogene Klartextdaten werden aus `<input_file>` ausgelesen. Aus diesen wird pro Datensatz mittels asymmetrischer hybrider Verschlüsselung ein Chiffrat erzeugt, wobei ein RSA–Schlüssel (public key) mit Kennung `<id>` und Paßwort `<password>` verwendet wird. Die Chiffrate werden in `<output_file>` geschrieben. Die Klartextdaten müssen, falls sie in der Datenbank in separaten Attributen gespeichert werden (Name, Straße, Ort, ...), zuvor vom Benutzer zu einer Zeichenkette verknüpft worden sein. Um nach einer späteren Dechiffrierung wieder eine Zerlegung in die einzelnen Attribute durchführen zu können, sollten die konkatenierten Komponenten durch ein geeignetes Trennzeichen separiert sein. Ein beispielhafter Klartextdatensatz koennte wie folgt aussehen:

```
Hans#Meier#Dorfstr. 12#12345 Kleckersdorf
```

Eingabeformat (F4)

```
<ID>
<Konkatenierter Klartext>
...
```

Ausgabeformat (F5)

```
<ID>
<Asymmetrisch verschluesseltes Chiffrat>
...
```

Hybride asymmetrische Entschlüsselung

```
UNICONFI e <input_file> <output_file> <id> <password>
```

Umkehrfunktion von `UNICONFI h ...`: Chiffrate werden aus `<input_file>` ausgelesen. Durch asymmetrische hybride Dechiffrierung erhält man die personenbezogenen Klartextdaten zurück, wobei ein RSA–Schlüssel (secret key) mit Kennung `<id>` und Paßwort `<password>` verwendet wird (muß zu dem public key gehören, der zum Verschlüsseln benutzt wurde). Die Klartextdaten werden in `<output_file>` geschrieben. War vor der Verschlüsselung eine Konkatenierung von Einzelattributen vorgenommen worden, kann diese nun rückgängig gemacht werden.

Eingabeformat: siehe F5

Ausgabeformat: siehe F4

Überverschlüsselung der baden–württemberger Kontrollnummern

```
UNICONFI b <input_file> <output_file> <id> <password>
```

Sonderfunktion für Baden–Württemberg: Baden–württemberger Kontrollnummern werden aus `<input_file>` ausgelesen. Durch eine Überverschlüsselung mit einem (bundesweit einheitlichen) IDEA–Schlüssel mit Kennung `<id>` und Paßwort `<password>` werden die Kontrollnummern für einen länderübergreifenden Abgleich nutzbar gemacht. Die überverschlüsselten Kontrollnummern werden in `<output_file>` geschrieben.

Eingabeformat (F6)

```
<ID>
<KN Baden-Wuerttemberg 1 als 16-stellige Ziffernfolge>
<KN Baden-Wuerttemberg 2 als 16-stellige Ziffernfolge>
...
```

Ausgabeformat (F7)

```
<ID>
<KN Baden-Wuerttemberg 1 ueberverschluesselt>
<KN Baden-Wuerttemberg 2 ueberverschluesselt>
...
```

3.3.2 UNICON Key Manager

Im folgenden werden die Aufrufparameter des UNICON Key Manager (Dateiname unter MS–DOS und Windows 3.1: UNICONKM.EXE, sonst Unicon-KeyManager) erläutert. Dieser bietet verschiedene Funktionen zur Verwaltung von IDEA– und RSA–Schlüsseln an, die vom UNICON File Interface benutzt werden.

```
UNICONKM i <id> <password>
```

Es wird ein mit `<password>` geschützter IDEA–Schlüssel mit Kennung `<id>` generiert.

```
UNICONKM r <id> <password> <len>
```

Es wird ein mit `<password>` geschütztes RSA–Schlüsselpaar (public und secret key) mit Kennung `<id>` und Länge `<len>` Byte generiert.

```
UNICONKM d <id> <password> <mode>
```

Der mit `<password>` geschützte RSA–Public– (`<mode> = 0`), RSA–Secret– (`<mode> = 1`) bzw. IDEA–Key (`<mode> = 2`) mit Kennung `<id>` wird gelöscht.

```
UNICONKM c <id> <old_password> <new_password> <mode>
```

Das Paßwort des RSA–Public– (`<mode> = 0`), RSA–Secret– (`<mode> = 1`) bzw. IDEA–Keys (`<mode> = 2`) mit Kennung `<id>` wird von `<old_password>` nach `<new_password>` geändert.

Schlüsseldatei

Die vom UNICON Key Manager verwalteten Schlüssel werden in einer ASCII–Datei persistent gespeichert. Name und Pfad dieser Datei sind prinzipiell frei wählbar, müssen jedoch in den Umgebungsvariablen `CACRYM_KEY_FILE` bzw. `CACRYM_KEY_PATH` abgelegt sein. Diese werden zur Laufzeit abgefragt. Bei `CACRYM_KEY_PATH` ist zu beachten, daß bei der Angabe von Verzeichnisnamen keine Backslashes, sondern nur Slashes verwendet werden und daß die Pfadangabe mit einem Slash enden muß. Ist `CACRYM_KEY_FILE` nicht gesetzt, wird per Default `key.dat` angenommen. `CACRYM_KEY_PATH` sollte auf jeden Fall gesetzt werden, und zwar auf eine absolute Pfadangabe wie z. B. `/krebsregister/unicon/`. Relative Pfadangaben wie `./` sind nicht zu empfehlen.

Die Schlüsseldatei kann beliebig viele Schlüssel aufnehmen. Sie enthält pro Schlüssel einen Datensatz, wobei zwei Datensätze durch eine Zeile mit „=“–Zeichen getrennt sind. Diese Zeile dient lediglich der besseren Übersicht. Jeder Datensatz besitzt folgenden Aufbau:

```
<Schluesselkennung>
<MD5-chiffriertes Passwort>
<Schluesseltyp>
<Schluessellaenge>
<Mit Passwort IDEA-verschluesselter Schluessel>
```

Mit der Schlüsselkennung kann ein Schlüssel eindeutig identifiziert werden. Ein Benutzer kann nur dann auf einen Schlüssel zugreifen, wenn er das zu einer Kennung passende Paßwort kennt. Dieses ist MD5–chiffriert in der Datei abgelegt. Des weiteren enthält der Datensatz Schlüsseltyp (`KEY_RSA_PK = 0, KEY_RSA_SK = 1, KEY_IDEA = 2`) und –länge. Schließlich ist der Schlüssel selbst gespeichert, allerdings aus Sicherheitsgründen IDEA–verschlüsselt mit dem (auf 16 Byte gekürzten oder verlängerten) Paßwort als Schlüssel.

Es folgt ein Beispiel für eine Schlüsseldatei mit zwei IDEA–Schlüsseln. RSA–Schlüssel können zwar auch gespeichert werden, sind jedoch für die Kontrollnummergenerierung nicht relevant.

```
========================================
unicon
@'o(<N;)*##WQFZ^?4=@x16
2
33
!lN9gM$mUAaZZf7nljHX4,$fMl2IUsx24
========================================
```

```
test
;\#Kk/o^tl;q!<T*m\=‘x16
2
33
?5c/PVa4YR2>+YV&]%KF9‘BU9[ZD%sx24
```

Mit dem Paßwort `gna3rz8` kann der Benutzer auf den Schlüssel mit der Kennung `unicon` zugreifen.

Kapitel 4

Weitere Aspekte

4.1 Implementierungsaspekte

Das UNICON–Modul ist für die Betriebssysteme Windows 95/NT, Windows 3.11, MS–DOS 6.22, Solaris 2.5 (UNIX) sowie SCO OpenServer 5.0.4 (UNIX) verfügbar. Es wurde in der objektorientierten Programmiersprache C++ implementiert. Um die UNICON–Routinen aus Fremdapplikationen heraus aufrufen zu können, wurde die Funktionsschnittstelle in ANSI–C realisiert. Die Entwicklung der Windows 95/NT–Version erfolgte mit dem Microsoft Visual C++ Compiler in der Version 5.0, für Windows 3.11 und MS–DOS 6.22 wurde auf Microsoft Visual C++ 1.52 zurückgegriffen. Die Solaris–Version wurde mit dem Sun WorkShop Compiler in der Version 4.2 entwickelt und mit dem Qualitätssicherungstool Purify in der Version 3.2 getestet. Die SCO–Implementierung schließlich basiert auf dem SCO–eigenen C++–Compiler (CC).

Tabelle 4.1 zeigt, welche UNICON–Varianten (Funktionsschnittstelle als dynamische oder statische Bibliothek, Dateischnittstelle als Executable) für welche Betriebssysteme vorliegen. Die Festlegung einer Regelung für die langfristige Wartung der UNICON–Software (inkl. Portierung auf neue Plattformen) ist Aufgabe der Arbeitsgemeinschaft bevölkerungsbezogener Krebsregister in Deutschland (ABKD).

Betriebssystem	Dynamische Bibliothek	Statische Bibliothek	Executable
Windows 95, Windows NT	√		√
Windows 3.11		√	
MS–DOS 6.22			√
Solaris 2.5	√	√	√
SCO OpenServer 5.0.4		√	√

Tabelle 4.1: Unterstützte Betriebssysteme

Im Rahmen des UNICON–Projektes wurden verschiedene kryptographische Bibliotheken an das System angebunden und nach mehreren Kriterien evaluiert. Im einzelnen wurden die Bibliotheken CryptoManager++ (Universität Hildes-

heim), SecuDE 5.0 (GMD Darmstadt) und SSLeay (E. A. Young) untersucht. Eine Verwendung des CryptoManager++ kam aufgrund des nicht gesicherten Supports und der geringen Verbreitung nicht in Frage. Gegen SecuDE 5.0 sprach, daß der Quellcode nicht frei verfügbar ist und nicht für alle relevanten Plattformen Bibliotheken vorliegen. Letztendlich fiel die Entscheidung zugunsten SSLeay aus, da hier der Quellcode frei vorliegt, eine einfache Portierbarkeit gewährleistet ist sowie ein hoher Verbreitungsgrad angenommen werden kann.

4.2 Rechtliche Aspekte

4.2.1 Vom BSI definierte Einsatzbedingungen

Über die im Software–Lizenzvertrag (siehe Anhang) festgeschriebenen Einsatzbedingungen hinaus hat das BSI eine Reihe von Einsatzbedingungen definiert, die den in UNICON enthaltenen Algorithmus für die Generierung der baden–württembergischen Kontrollnummern betreffen. Der Algorithmus wurde ursprünglich ausschließlich für das Krebsregister Baden–Württemberg vom BSI entwickelt und nun OFFIS zur Integration in UNICON überlassen.

Der Algorithmus darf, außer zu Testzwecken während der Entwicklungsphase bei OFFIS, nur in den Vertrauensstellen der Landeskrebsregister eingesetzt werden. OFFIS hat sich verpflichtet,

- alle Bestandteile des Algorithmus vertraulich zu behandeln,
- die Anzahl der mit dem Algorithmus betrauten Personen zu minimieren,
- den Algorithmus bzw. alle den Algorithmus enthaltende Programme ausschließlich an die Vertrauensstellen der Landeskrebsregister weiterzugeben,
- bei Weitergabe die Vertrauensstellen schriftlich über die Einsatzbedingungen (s. u.) zu informieren
- die Weitergabe dem BSI schriftlich mitzuteilen und
- nach Auslieferung des Algorithmus an die Vertrauensstellen den Algorithmus zu löschen und aus allen bei OFFIS verbleibenden Anwendungen zu entfernen.

Die Landeskrebsregister haben

- alle Bestandteile des Algorithmus vertraulich zu behandeln,
- den Algorithmus ausschließlich in den Vertrauensstellen aufzubewahren und einzusetzen und
- die Anzahl der mit dem Algorithmus betrauten Personen zu minimieren.

Zur Weitergabe des Algorithmus an Dritte, auch an andere Vertrauensstellen, bedarf es einer schriftlichen Genehmigung des BSI.

4.2.2 Klärung rechtlicher Fragen mit MatchWare

Seit Ende 1998 ist MatchWare Tochtergesellschaft von Vality Technology aus Boston. Bei Vality handelt es sich um einen etablierten Anbieter von Software–Tools zum Reengineering von Daten. Vom Zusammenschluß versprechen sich die beiden Firmen erhebliche Synergien in der Record Linkage–Technologie sowie in der Aneignung von applikationsspezifischem Know–How. Die Marktpräsenz der Firmen ist breit gefächert, insgesamt werden über 400 Lizenznehmer aus unterschiedlichen Branchen betreut (weitere Informationen unter http://www.vality.com). Vality hat den Support der MatchWare–Produkte übernommen. AutoMatch ist in das von Vality vertriebene Produkt Integrity integriert worden, wobei existierende AutoMatch–Applikationen auch innerhalb von Integrity unverändert lauffähig sind. Eine Upgrade–Möglichkeit von AutoMatch auf Integrity ist gegeben. Damit ist ein Support der Record Linkage–Software langfristig sichergestellt.

Der Hinterlegung des AutoMatch–Quellcodes bei einem Treuhänder steht MatchWare/Vality aufgeschlossen gegenüber. Eine endgültige Regelung dieser Frage steht jedoch noch aus. Hingegen hat sich MatchWare/Vality gegen den Vorschlag ausgesprochen, OFFIS zum Generalvertreter von MatchWare/Vality für die deutschen Krebsregister zu machen. Stattdessen möchte MatchWare/Vality direkt mit den Krebsregistern Geschäftsbeziehungen eingehen. Als gemeinsamer Ansprechpartner soll die ABKD (Sprecher: Joachim Schüz, Mainz) dienen.

Bei den Verhandlungen über günstigere AutoMatch–Lizenzgebühren für die deutschen Krebsregister konnte bisher keine Einigung erzielt werden. Dies hat in erster Linie mit der zögerlichen Haltung vieler Krebsregister bei der Frage nach der Anschaffung von AutoMatch zu tun, die wiederum in der angespannten finanziellen Situation der Register begründet ist. Interesse an AutoMatch haben Schleswig–Holstein, Münster, Hamburg, Bayern, Baden–Württemberg und das Robert–Koch–Institut bekundet, in Niedersachsen und Rheinland–Pfalz wird die Software bereits seit längerem eingesetzt. Im Zuge des Zusammenschlusses mit Vality Technology hat sich das Preisgefüge der MatchWare/Vality–Produkte leider nach oben verschoben. Die offizielle Lizenzgebühr für AutoMatch auf PC unter DOS, Windows95 oder Windows NT beträgt nun 8.000 US Dollar.

Kapitel 5

Einführung des Werkzeugs

5.1 Beteiligung der Krebsregister

Analog zur Spezifikation der Anforderungen an UNICON hatten die Landeskrebsregister auch bei der Einführung des Werkzeugs einen gewissen Eigenanteil zu leisten. Eine wesentliche Aufgabe bestand in der Mithilfe bei der Anbindung von UNICON an die im jeweiligen Register eingesetzte Applikation. Bei Verwendung der Dateischnittstelle mußten die vorgegebenen Ein– und Ausgabeformate berücksichtigt werden, bei Verwendung der Funktionsschnittstelle waren die Aufrufe der UNICON–Routinen in die Applikation zu integrieren. Nach Abschluß der Einführung konnte OFFIS davon profitieren, daß die Register ihre Erfahrungen mit dem Werkzeug mitteilten und Anregungen für Verbesserungen gaben.

Krebsregister	Variante	Version	Betriebssystem	Datum
Gemeinsames Krebsregister	D	1.0	MS–DOS	6/98
Rheinland–Pfalz	F	1.0	NT	6/98
Bayern	F	1.0	NT	7/98
Saarland	D	1.0	MS–DOS	8/98
Hamburg	D/F	1.0	SCO Unix	9/98
Niedersachsen	F	1.1	Solaris, NT	10/98
Bremen	D/F	1.1	NT	11/98
Schleswig–Holstein	D/F	1.1a	MS–DOS, NT	11/98
Münster	D	1.1a	MS–DOS	11/98
Baden–Württemberg	D/F	1.1a	MS–DOS, NT	11/98
Hessen	D/F	1.3	MS–DOS, NT	6/99

Tabelle 5.1: Einführung von UNICON

5.2 Verlauf der Einführung

Im 2. Quartal 1998 begann die Phase der Einführung von UNICON in den Landeskrebsregistern. Tabelle 5.1 zeigt, welches Register wann mit welcher

UNICON–Variante (Funktions– oder Dateischnittstelle) in welcher Version für welches Betriebssystem bedient wurde. Die Angabe zum Betriebssystem bezieht sich dabei nicht auf das tatsächlich installierte Betriebssystem, sondern auf die UNICON–Portierung. So kann z. B. die MS–DOS–Dateischnittstelle von UNICON auch unter NT eingesetzt werden. In der Tabelle wird nicht zwischen MS–DOS und Windows 3.11 unterschieden, ebenso wenig zwischen Windows 95 und NT.

Im Januar 1999 wurde die Version 1.2 von UNICON an alle Krebsregister verteilt. Hessen, Baden–Württemberg und Rheinland–Pfalz verfügen inzwischen über die aktuellste Version 1.3, die allerdings mit 1.2 kompatibel ist.

5.3 Lizenzverträge

Im Juni 1999 hat OFFIS Software–Lizenzverträge an die Landeskrebsregister verschickt. Darin sind die Einsatzbedingungen von UNICON zwischen OFFIS und den Krebsregistern geregelt. Die von den Krebsregistern unterschriebenen Verträge dienen weiterhin als Beleg für eine erfolgreiche Einführung der Software in den Ländern. Im Anhang sind ein Muster des Vertrags sowie die Unterschriften der Krebsregister in Kopie enthalten.

5.4 Offene Probleme

Im Rahmen der UNICON–Einführung sind einige kleinere Unzulänglichkeiten der Software zu Tage getreten. Aufgrund des relativ hohen Aufwandes, den eine wiederholte Neuinstallation der Software in allen Landeskrebsregistern mit sich bringen würde, hat sich OFFIS dazu entschieden, zunächst die Problembeschreibungen zu sammeln, die Landeskrebsregister darauf hinzuweisen und dann zu einem späteren Zeitpunkt ein neues Release von UNICON zu veröffentlichen, das alle bekannten Probleme behebt.

Bisher sind folgende Probleme gemeldet worden:

- Kurznamen wie „La“, die mit einem Füllsel übereinstimmen, werden von UNICON als Füllsel interpretiert, d. h. es wird eine Kontrollnummer für den 3. Namensteil gebildet, nicht jedoch für den 1. Namensteil. Dies könnte aus Sicht des Datenschutzes bedenklich sein, da aus dem Fehlen des 1. Namensteils geschlossen werden kann, daß der Name sehr kurz bzw. ein Füllsel sein muß. Des weiteren wird kein DDR–Namenscode gebildet, wenn der 1. Namensteil fehlt.

- Die 75–Variante der Kölner Phonetik liefert z. T. unterschiedliche Phonetik–Codes für Namen, die man intuitiv als gleich klingend bezeichnen würde. So wird z. B. der Name „Schröer“ auf „SREER“ abgebildet, während „Schroeer“ zu „SRER“ wird. Allgemein tritt das Problem bei Namen auf, bei denen auf einen Umlaut („ä“, „ö“ oder „ü“) der Buchstabe „e“ folgt (auch bei Verkettung mehrerer Namensteile). Hierbei handelt es sich nicht um einen Implementierungsfehler, sondern um eine Unzulänglichkeit der Kölner Phonetik an sich. Diese Unzulänglichkeit sollte beim

Abgleich von Meldungen berücksichtigt werden. Der Abgleich sollte sich nicht allein auf phonetische Codes abstützen, sondern auch noch andere Attribute zur Entscheidungsunterstützung heranziehen.

- Wird bei der Anwendung des hybriden RSA–Verfahrens ein ungültiger Schlüssel angegeben, so wird keine Fehlermeldung ausgegeben. Gleiches gilt, wenn die Schlüsseldatei mit einem Leerzeichen beginnt (falsches Dateiformat).

5.5 Vorbereitung eines bundesweiten Abgleichs

Um die Einsatzfähigkeit von UNICON bundesweit zu testen, hat die ABKD auf ihrer Sitzung im November 1998 beschlossen, die für einen testweisen bundesweiten Abgleich erforderlichen Daten an das Robert–Koch–Institut (RKI) zu schicken. Das RKI hat OFFIS gebeten, für diesen Abgleich einen bundeseinheitlichen IDEA–Schlüssel zur Überverschlüsselung der Kontrollnummern zu erzeugen und an die Krebsregister weiterzuleiten. Dieser Bitte ist OFFIS im März 1999 nachgekommen. Schlüssel und Paßwort wurden den Registern getrennt übermittelt. Das BSI hat darauf hingewiesen, daß der bundeseinheitliche IDEA–Schlüssel ausschließlich den Vertrauensstellen bekannt sein dürfe. In Zukunft solle ein solcher Schlüssel daher von einer ausgewählten Vertrauensstelle erzeugt und vertraulich an die anderen Vertrauensstellen weitergeleitet werden.

Literaturverzeichnis

[AMST96] H.-J. Appelrath, J. Michaelis, I. Schmidtmann, and W. Thoben. Empfehlung an die Bundesländer zur technischen Umsetzung der Verfahrensweisen gemäß Gesetz über Krebsregister (KRG). *Informatik, Biometrie und Epidemiologie in Medizin und Biologie*, 27(2):101–110, 1996.

[App96] H.-J. Appelrath. Projektvorschlag „Bundesweite Einführung eines einheitlichen Record Linkage–Verfahrens in den Krebsregistern der Bundesländer nach dem KRG“, 1996.

[Bun94] Deutscher Bundestag. Gesetz über Krebsregister (Krebsregistergesetz KRG). *Bundesgesetzblatt*, I(79):3351–3355, 1994.

[Jar89] Matthew A. Jaro. Advances in Record-Linkage Methodology as Applied to Matching 1985 Census of Tampa, Florida. *Journal of the American Statistical Association (JASA)*, 84(406):414–420, 1989.

[Jar94] M. A. Jaro. *AutoMatch — Generalized Record Linkage System — Version 2.9c — Getting Started*, 1994.

[Kol75] *Handbuch der medizinischen Dokumentation und Datenverarbeitung*, Stuttgart, 1975.

[Pos69] H.-J. Postel. Die Kölner Phonetik - Ein Verfahren zur Identifizierung von Personennamen auf Grundlage der Gestaltanalyse. *IBM-Nachrichten*, 19:925–931, 1969.

[Sch94] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, 1994.

[SM94] I. Schmidtmann and J. Michaelis. Untersuchungen zum Record Linkage für das Krebsregister Rheinland-Pfalz. Technical report, Tumorzentrum Rheinland-Pfalz, Mainz, August 1994.

[TA95] W. Thoben and H.-J. Appelrath. Verschlüsselung personenbezogener und Abgleich anonymisierter Daten durch Kontrollnummern. In H.-H. Brüggemann and W. Gerhardt-Häckl, editors, *Verläßliche IT-Systeme*, volume 22 of *DuD-Fachbeiträge*, pages 193–206, Rostock, 1995. Vieweg Verlag.

Anhang A

Muster des Software–Lizenzvertrages

Anhang B

Unterschriften der Landeskrebsregister (soweit vorliegend)