CryptoManager – Eine intuitiv verwendbare Bibliothek für kryptographische Systeme

Thilo Baldin, Gerrit Bleumer, Ralf Kanne
Institut für Informatik, Universität Hildesheim

ABSTRACT

Der CryptoManager ist eine Softwarebibliothek für kryptographische Verfahren, die nach folgenden Entwurfszielen entwickelt wurde: Individuell skalierbare Sicherheit, intuitive Verwendbarkeit, Effizienz, Wartbarkeit und Erweiterbarkeit, reiches Angebot kryptographischer Mechanismen. Insbesondere sollen Kooperationen mehrerer kryptographischer Verfahren für den Anwendungsprogrammierer wie ein (integriertes) Verfahren benutzbar sein, und potentiell unbegrenzte Datenoperanden (Datenströme) sollen einfach zu verarbeiten sein. Eine stabile Implementierung liegt in THINK-Pascal für Apple Computer vor.

Die Softwarebibliothek wurde zum Teil innerhalb des EU-Projekts SEISMED (Secure Environment for Information Systems in MEDicine) entwickelt. Die erfolgreiche Integration in eine prototypische Anwendung für gesicherten Dateitransfer in einem Apple-Talk-Rechnernetz demonstriert ihre Funktionalität und Effizienz.

Die Weiterentwicklung des CryptoManager verfolgt vorrangig die Entwicklungsziele Portierbarkeit und objektorientierten Entwurf. Konkret entwickelt wird eine C++-Implementierung.

1. EINFÜHRUNG

Unter kryptographischen *Systemarten* im engeren Sinne versteht man üblicherweise Konzelationssysteme, digitale Signatursysteme und Authentikationssysteme (Authentikationscodes). Die effiziente Realisierung dieser Systemarten zeigt, daß Pseudozufallsbitfolgengeneratoren zur Schlüsselerzeugung sowie verschiedene Arten Hashfunktionen benötigt werden. Der Einsatz dieser Komponenten ist sicherheitsrelevant, so daß nur solche Pseudozufallsbitfolgengeneratoren und Hashfunktionen eingesetzt werden sollten, die nicht unsicherer sind als die kryptographische Systemart, zu deren Implementierung sie beitragen. Derartige Pseudozufallsbitfolgengeneratoren und Hashfunktionen werden im folgenden ebenfalls als kryptographische Systemarten bezeichnet.

Man kann kryptographische Systemarten unterscheiden nach dem *Dienst*, den sie erbringen können. Im folgenden werden alle kryptographischen Systemarten, die denselben Dienst erbringen, einer *Systemklasse* zugeordnet. Der Dienst legt die Implementierung einer Systemklasse im allgemeinen nicht eindeutig fest. Daher existieren für jede Klasse viele Systemarten und für jede Systemart wiederum viele Implementierungen, die sich hinsichtlich Sicherheit, Effizienz etc. unterscheiden. Die Menge der fünf oben genannten kryptographischen Systemklassen zusammen mit den Hilfsklassen symmetrischer und asymmetrischer Blockchiffren ist algorithmisch

abgeschlossen in dem Sinne, daß zur Implementierung der enthaltenen Systemarten keine weiteren Systemklassen benötigt werden.

Allgemeiner könnte man auch folgende interaktiven Kooperationen [3, 20] als kryptographische Systemklassen verstehen:

- Austausch und Verteilung kryptographischer Schlüssel
- Gemeinsamer Münzwurf
- Zero-Knowledge Beweise
- Zwei– und Mehrparteienberechnungen mit gegenseitigen Geheimhaltungsforderungen
- Byzantinische Übereinstimmung
- Digitaler Wertetransfer und -austausch

In der Praxis sind Hersteller, Wartungsfirmen, Zertifizierungsstellen und Kunden daran interessiert, die Vielfalt möglicher Systemarten und Implementierungen für die gewünschten Dienste geeignet einzuschränken. Dies kann durch ein modulares Bausteinkonzept und Standardisierung der Bausteine geschehen. Die Wiederverwendbarkeit optimierter und getesteter Bausteine senkt die Herstellungskosten und vereinfacht die Wartung und Zertifizierung kompletter Applikationen, die kryptographische Verfahren verwenden. Ein gutes Bausteinkonzept unterstützt den Kunden z.B. bei der Anpassung seines Systems an neue Sicherheitsanforderungen.

Die Bausteine können wie folgt eingeteilt werden:

- **Arithmetiken** auf großen, endlichen Gruppen bzw. Ringen (Restklassenringen, Polynomringen, elliptischen Kurven, etc.),
- **Kryptographische Primitive**: Berechnungen, die von jedem Partner einer Kooperation lokal ausgeführt werden können, z.B. das Verschlüsseln einer Nachricht, sofern der Verschlüsselungsschlüssel zur Verfügung steht. Insbesondere auch das Generieren neuer kryptographischer Schlüssel.
- **Kommunikationsprimitve**: Operationen zum Senden und Empfangen von Daten. Sie sind abhängig von möglichen Adressierungsarten und vom verwendeten Betriebssystem.
- **Import- und Exportfilter**: Adapter, die auszutauschende Daten in das jeweils gewünschte Protokollformat umwandeln (X.400, X.500, EDI etc.).

Die vorliegende Bibliothek CryptoManager implementiert die fünf eingangs genannten Klassen kryptographischer Systeme, wobei der Schwerpunkt auf den kryptographischen Primitiven liegt, so daß weder Kommunikationsprimitve noch Filter angeboten werden. Sie bietet damit eine Bausteinbasis zur Implementierung allgemeinerer kryptographischer Systemklassen (s.o.).

Die Entwicklung der Bibliothek CryptoManager orientierte sich an der algorithmischen *Struktur* kryptographischer Systemarten. Kapitel 2 führt ein in die Konzepte des CryptoManager und geht auf Erfahrungen bei der Implementierung des CryptoManager 4.0 ein, die in THINK-Pascal vorliegt. Da kryptographische Systemarten von Beginn an als abstrakte Datentypen aufgefaßt wurden, war der nächste Schritt ein objektorientierter Entwurf des CryptoManager++. Dieser wird in Kapitel 3 vorgestellt. Kapitel 4 zeigt bisherige Einsatzbereiche und Erweiterungsmöglichkeiten des CryptoManager.

2. KONZEPTE UND IMPLEMENTIERUNG DES CRYPTOMANAGER 4.0

2.1 Kryptographische Systeme

Eines der wichtigsten Ziele bei der Konzeption des CryptoManager war die Bereitstellung einer *einheitlichen* und möglichst *intuitiv* zu verwendenden Programmierschnittstelle für *viele* unterschiedliche kryptographische Systemarten, die auf einfache Weise um neue Systemarten erweiterbar sein sollte. Aus diesem Grund wurde besondere Sorgfalt auf die Entwicklung eines geeigneten Konzeptes für die Realisierung kryptographischer Systemarten verwendet, mit dem sich die oben genannten Anforderungen erfüllen lassen.

2.1.1 Systemklassen und Systemarten

Kryptographische Systemarten lassen sich je nach Dienst klassifizieren (siehe Einleitung). Der CryptoManager bietet deshalb für kryptographische Systemarten vier[1] Systemklassen an[2]:

(1) *Konzelationssystem*

(2) *Signatursystem*

(3) *Pseudozufallsbitfolgengenerator*

(4) *Hashsystem*

Für jede Systemklasse stellt der CryptoManager verschiedene Systemarten bereit. Im einzelnen stehen zur Zeit (Version 4.0) die folgenden 16 Systemarten zur Verfügung:

Systemklasse Konzelationssystem

- **DES** : [7] (symmetrisch)
- **GDS** : Generalized DES [17] (symmetrisch)
- **IDA** : IDEA [14] (symmetrisch)
- **HYB** : Hybride Konzelation (s. z.B. [21])
- **RSA** : [19] (asymmetrisch)
- **ELG** : ELGAMAL [10] (asymmetrisch)
- **DMG** : DAMGÅRD [5] (asymmetrisch)

Systemklasse Signatursystem

- **GMR** : [12, 11]
- **RSS** : RSA Signatursystem [19].
- **EGS** : ELGAMAL [Elga_85]
- **DSS** : Digital Signature Standard [9]

Systemklasse Pseudozufallsbitfolgengenerator

- **PRG :** auf DES basierend
- **BBS** : x^2-mod-n Generator [1]

Systemklasse Hashsystem[3]

- **DGH** : [4] (asymmetrisch)
- **DSH** : [6] (schlüssellos)
- **RIP** : RIPE-MAC1 [18] (symmetrisch)

---

1 Da Signatursysteme im Vergleich zu Authentikationssystemen in der Praxis erweiterte Anwendungsmöglichkeiten besitzen, wird für sie im CryptoManager eine eigene Systemklasse angeboten. Authentikationssysteme sind zur Zeit nicht realisiert, eine Erweiterung um eine entsprechende Systemklasse ist aber leicht möglich.

2 Der Begriff *Systemklasse* im CryptoManager darf nicht mit dem Begriff *Klasse* in der objektorientierten Programmierung verwechselt werden.

3 In diese Klasse fallen auch die Hashsysteme MD4, MD5 und SHS (Secure Hash Standard), die im CryptoManager jedoch (noch) nicht implementiert sind.

Abb. 1 verdeutlicht die Einteilung in Systemklassen und Systemarten.

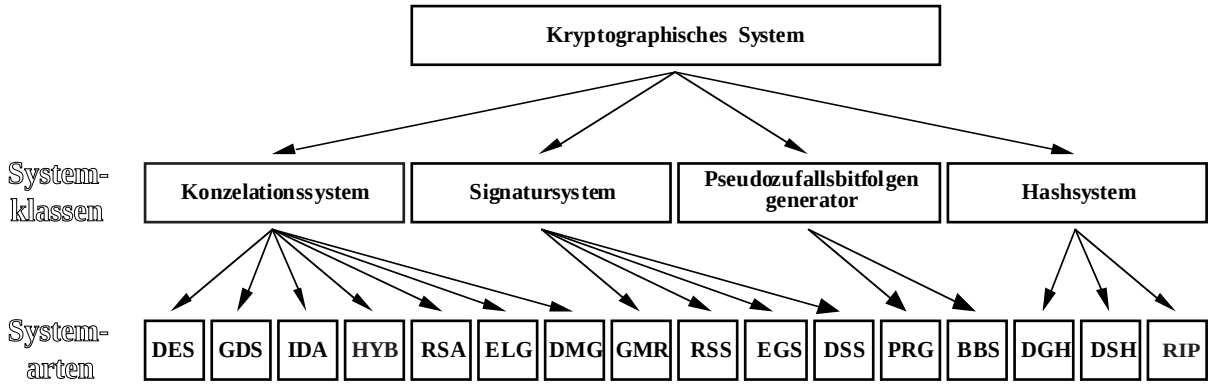

Abb. 1: Einteilung kryptographischer Systeme in Systemklassen und Systemarten

2.1.2 Deklaration durch abstrakte Datentypen

Alle Systemarten der angebotenen Systemklassen haben gemeinsam, daß sie aus einem Algorithmus zur Schlüsselgenerierung sowie aus einem oder zwei charakteristischen Algorithmen entsprechend der Systemklasse (i. allg. jeweils parametrisiert durch einen Schlüssel) bestehen. Es bietet sich daher an, eine solche *abstrakte* Sicht kryptographischer Systeme, die unabhängig von den konkreten Algorithmen ist, auf die Implementierung zu übertragen.

Der CryptoManager definiert für kryptographische Systemarten einen *universalen* Datentyp *CryptoSystem*, bestehend aus einem Wertebereich *Schlüssel* und mehreren Operationen. Operationen, die von den Systemarten jeder Systemklasse angeboten werden, heißen *universal*. Dazu gehören z.B. die Generierung sowie das Speichern und Laden von Schlüsseln. Operationen, die nur von Systemarten bestimmter Systemklassen angeboten werden, heißen *systemklassenspezifisch*, z.B. Verschlüsseln für die Systemklasse Konzelationssystem.

Neben dem abstrakten Datentyp *CryptoSystem* deklariert der CryptoManager auch jede Systemart als abstrakten Datentyp, wenn auch nur intern. Die Operationen des Datentyps *CryptoSystem* werden intern durch die Operationen der einzelnen Systemart-Datentypen realisiert (z.B. die Operation *Verschlüsseln* des Datentyps *CryptoSystem* durch die Operation *RSA-Verschlüsseln* für die Systemart RSA). Intern kann der CryptoManager daher leicht um neue Systemart-Datentypen erweitert oder existierende Systemart-Datentypen durch verbesserte Implementierungen ausgetauscht werden, ohne daß sich an dem externen Datentyp *CryptoSystem* (und damit für den Anwendungsprogrammierer) etwas ändert.

Es ist wichtig anzumerken, daß im Datentyp *CryptoSystem* die systemklassenspezifischen Operationen für die Systemarten aller Systemklassen vereinigt sind. Die systemklassenspezifischen Operationen von *CryptoSystem* werden für eine Systemart *A*, genau wie die universalen Operationen, (intern) durch die entsprechenden Operationen des Systemart-Datentyps von *A dynamisch emuliert*. Es erfolgt demnach keine wirkliche "Verfeinerung" des Datentyps *CryptoSystem*. Insbesondere werden die Systemklassen nicht als eigenständige abstrakte Datentypen deklariert, sondern dienen lediglich der Klassifizierung von Systemarten. Diese nicht ganz natürliche Modellierung hilft bei Verwendung einer nicht objektorientierten Programmiersprache (wie z.B. THINK-Pascal) darüber hinweg, daß keine Konzepte wie Vererbung und Operatorüberladung zur Verfügung stehen.

2.1.3 Kryptographische Systeminstanzen

Eine Instanzvariable des Datentyps *CryptoSystem* wird *kryptographische Systeminstanz* oder kurz *Systeminstanz* genannt. Jede kryptographische Systeminstanz ist eindeutig einer Systemart und damit eindeutig einer Systemklasse zugeordnet (vgl. Abb. 1). Um die Zuordnung einer Systeminstanz *I* zur Systemart *A* und Systemklasse *K* auszudrücken, wird *I* im folgenden als *kryptographische Systeminstanz der Systemart A* bzw. *kryptographische Systeminstanz der Systemklasse K* bezeichnet.

Für Systeminstanzen der Systemklasse *K* sind alle universalen Operationen des Datentyps *CryptoSystem* verfügbar sowie alle für *K* systemklassenspezifischen Operationen. Wie bereits erwähnt, werden die universalen und systemklassenspezifischen Operationen des Datentyps *CryptoSystem* für eine kryptographische Systeminstanz der Systemart *A* beim Aufruf durch die entsprechenden Operationen des (internen) Datentyps der Systemart *A* dynamisch emuliert. Der Aufruf einer systemklassenspezifischen Operation mit einer unpassenden Systeminstanz führt zu einem Fehler. Für kryptographische Systeminstanzen stehen im CryptoManager u.a. die folgenden Operationen zur Verfügung:

Universale Operationen

- Anlegen (dynamische Allozierung) einer kryptographischen Systeminstanz einer gewünschten Systemart
- Generierung eines vollständigen Instanzschlüssels (geheimer bzw. öffentlicher und privater)
- Speichern eines geheimen bzw. öffentlichen bzw. öffentlichen und privaten Instanzschlüssels als externe Bytefolge
- Laden eines als externe Bytefolge gespeicherten Instanzschlüssels

Systemklassenspezifische Operationen

- Erzeugung von Pseudozufallsbitfolgen gewünschter Länge mit einer Systeminstanz der Systemklasse Pseudozufallsbitfolgengenerator
- Hashen von Nachrichten mit einer Systeminstanz der Systemklasse Hashsystem
- Ver- bzw. Entschlüsselung von Klar- bzw. Schlüsseltexten mit einer Systeminstanz der Systemklasse Konzelationssystem
- Erzeugung bzw. Verifikation digitaler Signaturen für Nachrichten mit einer Systeminstanz der Systemklasse Signatursystem

2.1.4 Kombinierte Systemarten

Häufig ist es erforderlich, Operationen einer Systemart mit denen einer anderen zu kombinieren:

- Bei der Erzeugung bzw. Verifikation einer Signatur wird auf die jeweilige Nachricht im allgemeinen zunächst eine kollisionsfreie Hashfunktion angewendet.
- Die Realisierung von Redundanzprädikaten für das RSA-Konzelationssystem basiert ebenfalls auf kollisionsfreien Hashfunktionen.
- Bei einem hybriden Konzelationssystem wird sowohl ein symmetrisches Konzelationssystem für die Verschlüsselung des Klartextes als auch ein asymmetrisches Konzelationssystem für die Verschlüsselung des jeweiligen symmetrischen Schlüssels benötigt.

In solchen Fällen ist es sinnvoll, die beteiligten Systemarten in einer Gesamtsystemart zusammenzufassen und die kombinierte Anwendung von Operationen der verschiedenen Systemarten

als einheitliche Operationen der Gesamtsystemart zur Verfügung zu stellen. Auf diese Weise kann die gleichzeitige Verwaltung der verschiedenen beteiligten Schlüssel für den Anwendungsprogrammierer vereinfacht und die Transparenz der bereitgestellten Operationen erhöht werden.

Im CryptoManager werden aus verschiedenen Systemarten zusammengesetzte Systemarten *kombinierte Systemarten* genannt. Kombinierte Systemarten unterscheiden sich intern von den bis jetzt bekannten *atomaren Systemarten* durch den Aufbau der jeweiligen kryptographischen Systeminstanzen, stehen an der Schnittstelle jedoch als *einheitliche* Systemarten einer entsprechenden Systemklasse zur Verfügung. Eine kombinierte Systemart besteht aus einer *Obersystemart*, nach der auch die kombinierte Systemart benannt wird, und einer oder mehreren *Untersystemarten*, die selbst wieder Obersystemarten anderer Untersystemarten sein können.

Kombinierte Systemarten sind RSS, EGS und DSS (Systemklasse Signatursystem) sowie RSA (Systemklasse Konzelationssystem) mit jeweils einer Untersystemart der Systemklasse Hashsystem. Hierbei wurden die entsprechenden Hashoperationen (s.o.) in die Signier- bzw. Verschlüsselungsoperationen integriert . Die Systemart HYB (Systemklasse Konzelationssystem) ist ebenfalls eine kombinierte Systemart, die aus einer asymmetrischen und einer symmetrischen Systemart der Systemklasse Konzelationssystem besteht, deren kombinierter Einsatz in den Verschlüsselungsoperationen der hybriden Gesamtsystemart ebenfalls zusammengefaßt wurde. Tab. 1 gibt einen Überblick über kombinierte Systemarten im CryptoManager[4].

Systemart	Systemklasse	Anzahl Untersystemarten	Mögliche Systemarten 1. Untersystemart	Mögliche Systemarten 2. Untersystemart	Systemklasse der Untersystemarten
RSS	Signatursystem	1	DGH, DSH, RIP	—	Hashsystem
EGS	Signatursystem	1	DGH, DSH, RIP	—	Hashsystem
DSS	Signatursystem	1	DGH, DSH, RIP	—	Hashsystem
RSA	Konzelationssystem	1	DGH, DSH, RIP	—	Hashsystem
HYB	Konzelationssystem	2	RSA, ELG, DMG	DES, GDS, IDA	Konzelationssystem

Tab.1: Überblick über kombinierte Systemarten des CryptoManager

2.2 Dynamische Datenoperanden und deren Verarbeitung

2.2.1 Dynamische Datenoperanden

Die meisten Datenoperanden des CryptoManager sind dynamisch, d.h. ihre Länge ist zur Übersetzungszeit nicht festgelegt. Dazu gehören u.a.

- extern gespeicherte kryptograhische Schlüssel
- Pseudozufallsbitfolgen
- Nachrichten / Klartexte
- Schlüsseltexte
- Hashwerte
- Digitale Signaturen

4 Für eine kombinierte Systemart *A* legt der CryptoManager lediglich für jede Untersystemart die Systemklasse fest, so daß alle Systemarten dieser Systemklasse als jeweilige Untersystemart einer anzulegenden Systeminstanz der Systemart *A* gewählt werden können. (Vergleiche auch Kap. 3.3.) Für die konkrete Wahl muß der Anwendungsprogrammierer zwischen den unterschiedlichen Sicherheits- und Effizienzeigenschaften der zur Verfügung gestellten Systemarten abwägen.

Ein dynamischer Operand wird im CryptoManager durch eine typlose Referenz auf den Operandenanfang und die Operandenlänge spezifiziert, besteht also aus einem Tupel (a, l), wobei a die Anfangsadresse und l die Länge des Operanden bezeichnet. Dynamische Operanden werden demnach als zusammenhängende Bytefolgen manipuliert, die von Routinen des CryptoManager als solche aus entsprechenden Speicherbereichen gelesen bzw. in entsprechende Speicherbereiche geschrieben werden. Eine solche Repräsentation dynamischer Operanden hat den Vorteil, daß sie elementar ist und keine weitere Typdeklaration erfordert.

2.2.2 Vollständige und unvollständige Operanden

Manche Operationen des CryptoManager sollen Eingaben erlauben, deren Länge die Größe des zur Verfügung stehenden Arbeitsspeichers möglicherweise übersteigt. Dies kann z.B. ein auf einem externen Datenträger gespeicherter langer Klartext sein. Der CryptoManager bietet dem Anwendungsprogrammierer in diesem Fall die Möglichkeit, eine Eingabe $in = (a, l)$ in Teiloperanden (*Cluster*) $c_1 = (a_1, l_1)$,..., $c_n = (a_n, l_n)$ zu zerlegen und die einzelnen Cluster c_1,..., c_n sequentiell zu verarbeiten[5]. Cluster stellen ebenfalls dynamische Operanden dar, deren Eigenschaften sich aber von denen anderer Operanden in mancher Hinsicht unterscheiden. Der CryptoManager differenziert deshalb zwischen zwei Arten von dynamischen Operanden:

(1) *vollständige Operanden*

(2) *unvollständige Operanden* (Cluster)

Ein Operand heißt vollständig, wenn er grundsätzlich eine komplette Eingabe ist. Bei einem unvollständigen Operanden handelt es sich *potentiell* nur um einen Teil einer Eingabe. Analog unterscheidet der CryptoManager auch bei Ausgaben vollständige und unvollständige Operanden.

Für Nachrichten bzw. Klartexte und Schlüsseltexte bietet der CryptoManager Clusterzerlegung und entsprechend Operationen, die unvollständige Operanden (Nachrichten-, Klartext- bzw. Schlüsseltextcluster) erlauben. Extern gespeicherte Schlüssel, Pseudozufallsbitfolgen, Hashwerte und Signaturen sind vollständige Operanden.

Für die Aufnahme eines Ergebnisoperanden muß der Anwendungsprogrammierer grundsätzlich außerhalb des CryptoManager vor der Operationsausführung einen ausreichend großen Speicherbereich allozieren (auf diese Weise können Operanden direkt ab einer vom Programmierer spezifizierten Adresse geschrieben werden). Für vollständige Operanden (extern gespeicherte Schlüssel, Hashwerte oder Signaturen) stellt der CryptoManager Routinen zur Verfügung, mit deren Hilfe der Anwendungsprogrammierer den für einen Ergebnisoperanden *mindestens* zu allozierenden Speicher bestimmen kann. Die Größe eines für einen unvollständigen Ergebnisoperanden (Klartext- oder Schlüsseltextcluster) zu allozierenden Speicherbereichs entspricht der maximalen Clusterlänge (siehe Kap. 2.2.3).

---

[5] Der Anwendungsprogrammierer muß die Zerlegung einer Eingabe in Cluster aus Gründen der Flexibilität (z.B. für den Einsatz in höheren Protokollen) explizit vornehmen können. Dafür bietet der CryptoManager komfortable Hilfsroutinen an, so daß der Anwendungsprogrammierer keinerlei Berechnungen mehr durchführen muß (siehe Kap. 2.2.3). Die ebenfalls zur Verfügung stehende Bibliothek *SmartFileManager* wendet diese Hilfsroutinen an und erweitert die Funktionalitäten des CryptoManager auf Dateien. Dieser gestattet, Dateien als Ganzes zu verarbeiten, ohne daß der Anwendungsprogrammierer sich um Clusterzerlegungen kümmern muß.

2.2.3 Clusterkonzept

Die Verarbeitung einer Eingabe, die als Ganzes nicht im Hauptspeicher gehalten und manipuliert werden kann, gliedert sich bei Verwendung des CryptoManager in die folgenden drei Schritte:

(1) Bestimmung einer Clusterzerlegung der Eingabe.

(2) Sequentielle Verarbeitung der Cluster mittels einer vom CryptoManager zur Verfügung gestellten kryptographischen Operation.

(3) Zusammensetzen der Ergebniscluster zu einer Ausgabe (i.allg. mit Schritt (2) verzahnt).

Die Schritte (1) und (2) sind grundsätzlich erforderlich, während Schritt (3) nur dann durchgeführt werden muß, wenn die Operationsergebnisse ebenfalls unvollständige Operanden sind (resultierende Schlüsselt- bzw. Klartextcluster bei einer Ver- bzw. Entschlüsselungsoperation).

Die *maximale Länge* eines Clusters wird nur durch die Größe des Hauptspeichers begrenzt und kann vom Anwendungsprogrammierer frei gewählt werden. Nach Festlegung der maximalen Clusterlänge müssen für ein Eingabe- und ein Ergebniscluster zwei entsprechend große Speicherbereiche (Eingabe- und Ergebnispuffer) alloziert werden. In den Eingabepuffer kann dann ein Cluster einer Eingabe z.B. von der Platte geladen und von dort durch eine entsprechende Routine des CryptoManager eingelesen und verarbeitet werden. Der Ergebnispuffer dient zur Aufnahme des evtl. resultierenden Ergebnisclusters, das anschließend z.B. wieder auf die Platte gespeichert werden kann. Die sequentielle Verarbeitung von Clustern unter Verwendung eines Eingabe- und Ergebnispuffers verdeutlicht Abb. 2.

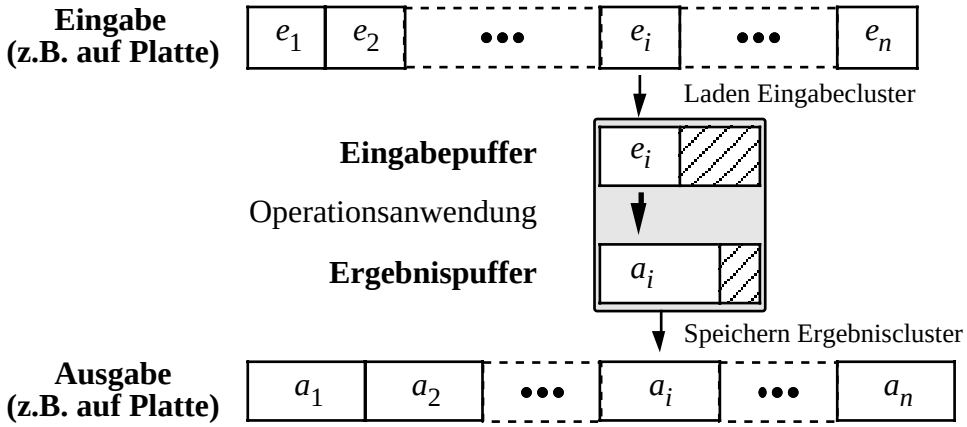

Abb. 2: Sequentielle Verarbeitung von Clustern

Die Längen der einzelnen Cluster c_i einer Clusterzerlegung $c_1,.., c_n$ dürfen nicht immer der festgesetzten maximalen Clusterlänge entsprechen. So darf z.B. die Länge eines Eingabeclusters nur so klein gewählt werden, daß das resultierende Ergebniscluster noch von einem Speicherbereich der maximalen Clusterlänge aufgenommen werden kann. Es zeigt sich, daß die Längen der einzelnen Cluster c_i in Abhängigkeit von der darauf anzuwendenden Operation, der dafür zu verwendenden Systeminstanz, von deren Systemart und von der Clusterposition i (wobei eine Unterscheidung von „ersten“, „mittleren“ und „letzten“ Clustern ausreicht) zu berechnen sind. Der Crypto Manager unterstützt deshalb die Zerlegung einer Eingabe in Cluster durch Routinen, die für eine beliebige Eingabe in Abhängigkeit von den zuvor aufgeführten Parametern Clusterlängen und die daraus resultierende Clusteranzahl berechnen. Auf diese Weise wird der Anwendungsprogrammierer von der (nicht trivialen) Clusterlängenberechnung für die Zerlegung einer Eingabe befreit.

2.3 Individuelle Konfiguration kryptographischer Systeminstanzen

Der CryptoManager bietet die Möglichkeit, Systeminstanzen durch Wahl der sicherheitsrelevanten Parameter *individuell* zu konfigurieren, um dem Anwendungsprogrammierer größtmögliche Flexibilität bei der Anpassung der kryptographischen Operationen an das mit seiner Anwendung verbundene Risiko zu erlauben. Zur Erhöhung der Praktikabilität können Konfigurationsparameter sowohl für *alle* Systeminstanzen einer Systemart als auch für *einzelne* Systeminstanzen gewählt werden.

2.3.1 Konfigurationsparameter

Abhängig von der Systemart erlaubt der CryptoManager für kryptographische Systeminstanzen die Wahl der folgenden Konfigurationsparameter:

- Modullängen für kryptographische Systeminstanzen auf modularer Arithmetik basierender Systemarten.
- Untersystemarten für Systeminstanzen kombinierter Systemarten.
- Betriebsarten (ECB, CBC, PCBC) für Systeminstanzen der Systemarten DES, GDS und RSA.
- Redundanzprädikate für Systeminstanzen der Systemart RSA.

2.3.2 Globale und lokale Konfiguration

Die *globale Konfiguration* des CryptoManager bestimmt, welche Konfigurationsparameterwerte bei der Schlüsselgenerierung für kryptographische Systeminstanzen einer bestimmten Systemart verwendet werden. Der CryptoManager stellt dafür eine *globale Konfigurationstabelle* bereit, die die Konfigurationswerte für jede Systemart enthält, und die bei der Initialisierung des CryptoManager mit entsprechenden Default-Werten aus einer Resource-Datei geladen wird. Die Einträge in der globalen Konfigurationstabelle für eine Systemart *A* legen somit die bei der Schlüsselgenerierung für alle Systeminstanzen der Systemart *A* zu verwendenden Konfigurationswerte fest. Sie können jedoch vom Anwendungsprogrammierer und, falls dieser dies für seine Anwendung vorsieht, vom Endanwender zur Laufzeit jederzeit geändert werden. Auf diese Weise ist prinzipiell jeder Schlüssel einer Systeminstanz der Systemart *A* mit einer individuellen Konfiguration erzeugbar.

Die bei der Schlüsselgenerierung für eine kryptographische Systeminstanz *I* verwendeten globalen Konfigurationswerte werden als *lokale Konfiguration* der Systeminstanz *I* bezeichnet. Für Systeminstanzen einiger Systemarten lassen sich bestimmte lokale Konfigurationsparameter (z.B. das zu verwendende Redundanzprädikat bzw. die zu verwendende Betriebsart) auch nach der Schlüsselgenerierung unabhängig von der globalen Konfiguration noch ändern.

2.4 Laufzeitergebnisse

Die Implementierung des CryptoManager 4.0 erfolgte in THINK Pascal und mit einer überwiegend in Motorola 680x0 Assembler geschriebenen Langzahlarithmetik. Messungen auf einem Apple Quadra 950 (MC68040, 30 MHz, 256 Byte Befehls-Cache, 256 Byte Daten-Cache) ergeben z.B. Verschlüsselungsgeschwindigkeiten von etwa 1,4 MBit/s für DES und 1,8 MBit/s für

IDEA. Für nicht-hybrides RSA werden bei einer Modullänge von 512 Bit etwa 50 KBit/s beim Verschlüsseln und etwa 5 KBit/s beim Entschlüsseln erreicht.

3. OBJEKTORIENTIERTE KONZEPTION DES CRYPTOMANAGER++

Zur Zeit wird der CryptoManager++ als portable objektorientierte Klassenbibliothek unter der Programmiersprache C++ entwickelt. Der CryptoManager++ realisiert alle grundlegenden Konzepte des CryptoManager 4.0, verfeinert jedoch das Konzept der Systemklassen und Systemarten, indem die kryptographischen Systeme zusätzlich nach ihrer Schlüsselstruktur unterschieden werden.

3.1 Ziele

Es wird angestrebt, den CryptoManager++ so zu gestalten, daß er möglichst einfach auf mehrere Plattformen — SUN, Apple-Macintosh, PC — portiert werden kann. Allerdings steht diese Zielsetzung im Konflikt mit dem Ziel der Effizienz: Maximale Portierbarkeit läßt sich erreichen, wenn die Programmierung vollständig in einer genormten Hochsprache wie C++ erfolgt. Deutlich höhere Effizienz aber kann durch den geschickten Einsatz in Assemblersprache geschriebener Konstrukte erreicht werden. Im CryptoManager++ wird daher ein minimaler Assemblerkern angestrebt. Laufzeitergebnisse des CryptoManager++ liegen derzeit noch nicht vor.

Der objektorientierte Entwurf fördert die intuitive Verwendbarkeit des CryptoManager++. Alle verfügbaren Operationen einer Systemklasse können, unabhängig von der konkreten Systemart, ebenfalls unter dem gleichen Namen ausgeführt werden. Sie werden jedoch als Methoden nicht wie im CryptoManager 4.0 dynamisch emuliert sondern (statisch) deklariert. Insbesondere mit Hilfe der objektorientierten Vererbungstechnik lassen sich die Verhältnisse zwischen Systemklassen und Systemarten in einer natürlichen Weise modellieren, wie in den folgenden Unterkapiteln ausgeführt wird. Die Erweiterung um neue Systemarten ist damit problemlos möglich.

3.2 Objektklassen und Vererbung

Die Bausteine einer objektorientierten Architektur sind die sogenannten *Objektklassen*. Die Deklaration einer Objektklasse beschreibt die Operationen eines abstrakten Datentyps (ADT) durch öffentlich deklarierte (im folgenden kurz *öffentliche*) Methoden. Jede Instanz einer Objektklasse wird *Objekt* genannt. Der Zugriff auf die Daten eines Objekts erfolgt ausschließlich über die öffentlichen Methoden der zugehörigen Objektklasse (*information hiding*). Die Implementierungsdetails der Methoden werden verdeckt. Die Deklaration und Implementierung von Objektklassen kann inkrementell mit Hilfe der *Vererbungstechnik* aus anderen Objektklassen abgeleitet werden. So wird eine Vererbungshierarchie von Objektklassen aufgebaut, innerhalb der eine Objektklasse (Erbe) aus einer oder mehreren Objektklassen (Vorfahren) nicht-zyklisch abgeleitet wird. Dabei kann ein Erbe die Methoden der Vorfahren übernehmen, zur Spezialisierung ersetzen (überladen) sowie um neue Methoden ergänzen.

3.3 Der CryptoManager++ als Klassenbibliothek

Für die Vererbungshierarchie des CryptoManager++ (Abb. 3) bezeichne im folgenden die Vererbungsebene *i* die Menge aller Objektklassen mit Abstand *i* von der Wurzelklasse CBASE. CBASE enthält allgemeine Hilfsmethoden und –strukturen.

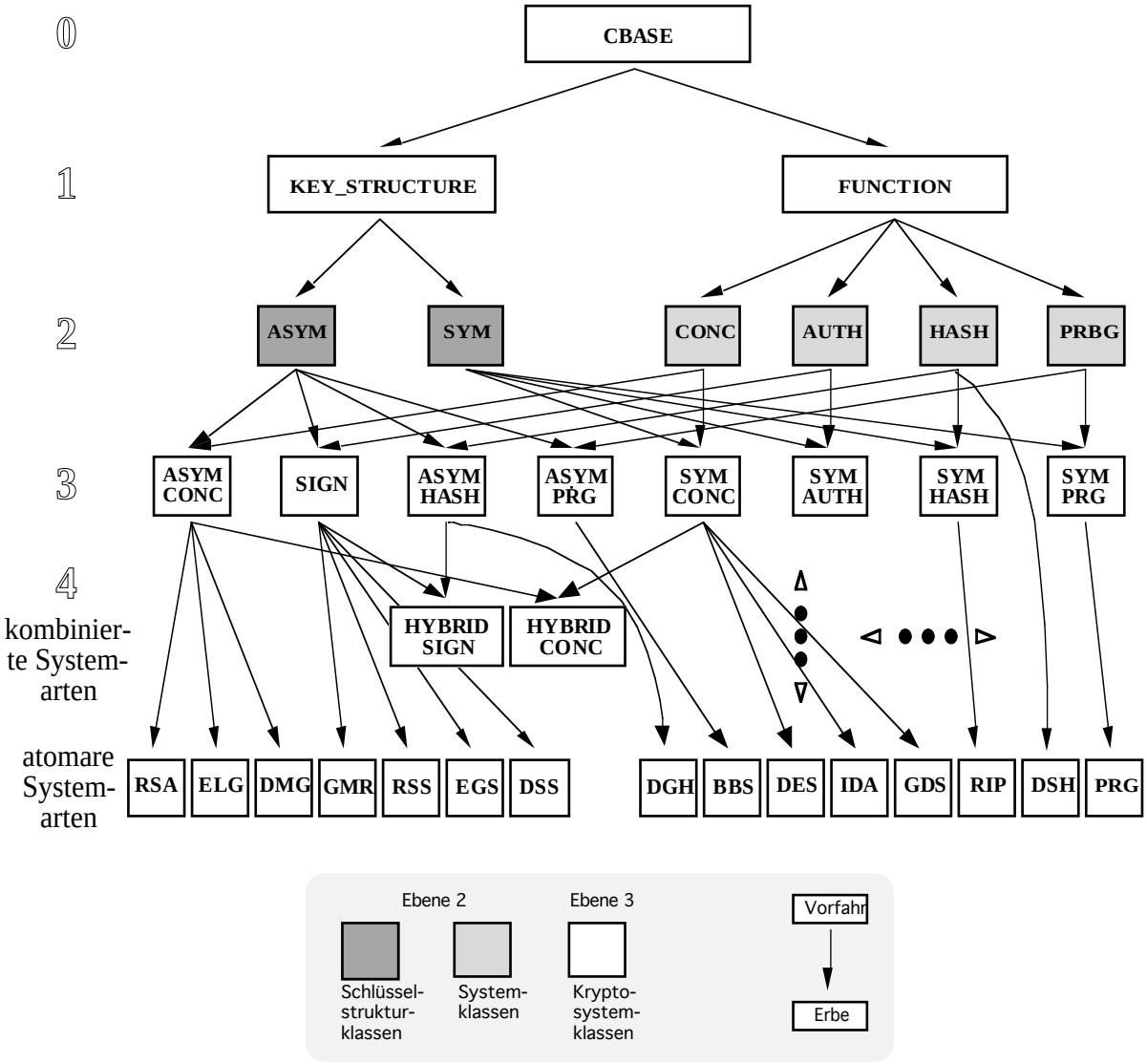

Abb. 3: Vererbungshierarchie des CryptoManager++

Auf Vererbungsebene 1 wird eine orthogonale Einteilung der kryptographischen Systeme nach Schlüsselstrukturen (Klasse KEY_STRUCTURE) sowie nach Funktionalitäten (Klasse FUNCTION) vorgenommen. Durch die Kombination von Schlüsselstruktur und Funktionalität läßt sich eine abstrakte Beschreibung aller betrachteten kryptographischen Systeme erreichen. Auf der Vererbungsebene 2 wird diese Einteilung verfeinert, indem zwei Schlüsselstrukturklassen ASYM (public key Verfahren) und SYM (one key Verfahren) sowie vier Systemklassen CONC (Konzelationssysteme), AUTH (Authentikationssysteme), HASH (Hashsysteme) und PRBG (Systeme zur Pseudozufallsbitfolgenerzeugung) als Ableitungen der Klassen KEY_STRUCTURE bzw. FUNCTION unterschieden werden. Diese stellen die jeweils spezifischen Operationen als öffentliche Methoden zur Verfügung (Tab. 2).

		Deklarierte, öffentliche Methoden
Schlüsselstruk-turklassen	**ASYM**	Generieren, Laden und Speichern von öffentlichen und privaten Schlüsseln
	SYM	Generieren, Laden und Speichern von geheimen Schlüsseln
System-klassen	**CONC**	Verschlüsseln und Entschlüsseln, Hilfsmethoden (z.B. zur Clustereinteilung)
	AUTH	Signieren und Signatur testen, Hilfsmethoden (z.B. zur Clustereinteilung)
	HASH	Hashen, Hilfsmethoden (z.B. zur Clustereinteilung)
	PRBG	Pseudozufallsbitfolge erzeugen

Tab. 2: Die öffentlichen Methoden der Schlüsselstruktur- und Systemklassen

Auf der Vererbungsebene 2 wird das Konzept der dynamischen Datenoperanden (Kap. 2.2) angewendet.

Aus jeweils einer Schlüsselstrukturklasse und einer Systemklasse werden auf der Vererbungsebene 3 acht Kryptosystemklassen abgeleitet, die die ADTen aller[6] betrachteten kryptographischen Systeme darstellen. Die Methoden sowohl der Schlüsselstrukturklasse als auch der Systemklasse werden geerbt und stehen automatisch in der Kryptosystemklasse zur Verfügung. Neue Methoden werden nicht hinzugefügt. Beispielsweise stellt die Kryptosystemklasse ASYM_CONC der asymmetrischen Konzelationssysteme an ihrer Schnittstelle die öffentlichen Methoden zum *Generieren, Laden* und *Speichern* von *öffentlichem* und *privatem* Schlüssel ebenso zur Verfügung, wie die öffentlichen Methoden zum *Verschlüsseln* und *Entschlüsseln.*

Auf den Vererbungsebenen 0 bis 3 können nur die für mehrere Systemarten identischen Methodenteile implementiert werden. Daher sind die Objektklassen dieser Vererbungsebenen nur virtuell definiert und können nicht instanziiert werden.

Kombinierte Systemarten verwenden gleichzeitig Schlüssel und Operationen mehrerer Systemarten. Aufgrund dieser Eigenschaft erfolgt in der Hierarchie die abstrakte Modellierung der kombinierten Systemarten in der Vererbungsebene 4 durch Ableitung aus (mehreren) Kryptosystemklassen. Die kombinierten Systemarten bedienen sich als Client der öffentlichen Methoden anderer Systemarten (Server). Dazu wird (in C++) in dem Client eine Instanz der Server angelegt. Die Realisierung dieses Client–Server Konzepts [15] erfolgt im CryptoManager++ sehr flexibel durch Deklaration der Client–Klasse als *template-class* (parametrisierte Klasse). Statisch festgelegt werden lediglich die Kryptosystemklassen der Server, während zur Laufzeit jede abgeleitete Systemart als Server ausgewählt werden kann. Dadurch stehen beispielsweise in der *template-class* HYBRID_CONC $3 \times 3 = 9$ Möglichkeiten zur Verfügung, ein hybrides Konzelationssystem zu instanziieren. Als Server kann jedes Paar einer Systemart von ASYM_CONC und einer Systemarten von SYM_CONC benutzt werden.

Außer den schlüssellosen Hashsystemen werden alle atomaren Systemarten als Erben jeweils genau einer Kryptosystemklasse realisiert. In ihnen werden die in den oberen Ebenen fehlenden Implementierungen der artspezifischen Methoden durchgeführt. Beispielsweise wird der RSA–Algorithmus (modulare Exponentiation) erst in der Objektklasse RSA implementiert. Dagegen werden die für alle Konzelationssysteme identischen Betriebsarten bereits in der Klasse CONC implementiert.

---

6 Eine Ausnahme sind die schlüssellosen Hashsysteme; ihr ADT wird bereits von der Systemklasse HASH dargestellt.

3.4 Erweiterbarkeit

Die Klassenhierarchie ist recht einfach um neue Systemarten erweiterbar, indem diese als Erben der entsprechenden Kryptosystemklasse(n) eingefügt werden. In der Klasse einer neuen atomaren Systemart müssen lediglich die artspezifischen Implementationen so vorgenommen werden, daß die Spezifikationen, die das Ein–/Ausgabeverhalten der Methoden von Systemarten beschreiben, erfüllt werden. Dann ist die korrekte Implementierung der Operationen im Zusammenspiel mit den Methoden der Vorfahrklassen gewährleistet. Auch kombinierte Systemarten sind leicht zu ergänzen, indem angegeben wird, welche Kryptosystemklassen oder schon vorhandenen Klassen kombinierter Systemarten als Vorfahren dienen sollen (Vererbungsabhängig-keit). Die eigentliche Implementierungsarbeit erfolgt in der Klasse der einzufügenden Systemart, indem aus den Methoden der Vorfahrklassen die Methoden der eingefügten kombinierten Systemart zusammengesetzt werden. In der Regel beeinflußt keine der vorgenommenen Erwei-terungen andere Objektklassen in der Hierarchie.

4. EINSATZ UND WEITERE ENTWICKLUNG DES CRYPTOMANAGER

In dem Maße, in dem komplexe IT-Systeme für sicherheitsrelevante Aufgaben eingesetzt werden, müssen sie erhöhte Ansprüche sowohl von *Beteiligten* als auch von *Betroffenen* erfüllen. Denn charakteristisch für diese Systeme sind komplexe Konstellationen des Vertrauens und Mißtrauens zwischen denen, die die Systeme entwerfen, anschließend warten und pflegen, sie aktiv benutzen, und von ihnen passiv betroffen sind. Als technische Realisierung individueller Sicherheitsanforderungen, die auch bei Anwesenheit anderer, mißtrauter Teilnehmer durchgesetzt werden sollen, sind kryptographische Verfahren unerläßlich. Deren Anwendbarkeit bei der Übertragung und Verarbeitung von Gesundheitsdaten wird u.a. im EU-Projekt SEISMED (Secure Environment for Information Systems in MEDicine) untersucht. Dafür ist am Institut für Informatik der Universität Hildesheim ein Prototyp SECURE Talk™ entwickelt worden, der benutzerfreundlichen, sicheren und effizienten Ende-zu-Ende Dateitransfer demonstriert [2]. Verschlüsselung, digitale Signaturen und ein transparentes Schlüsselmanagement öffentlicher Schlüssel werden mit Hilfe des CryptoManager realisiert. Zusammen mit der Universitätsklinik Leiden (Niederlande) wird an einer Integration des CryptoManager in kommunikationsreiche medizinische Anwendungen gearbeitet.

Ebenfalls eingesetzt wird der CryptoManager für ein Datensicherheitspraktikum, das an der Universität Hildesheim und der Technischen Universität Dresden im Hauptstudium der Informatik angeboten wird [16]. Hier hat sich der modulare Aufbau der Bibliothek auch unter didaktischen Aspekten bewährt, weil er eine Beschäftigung mit „dem Wesentlichen“ ermöglicht. Z.B. können die Systemarten-Module, die ja die kryptographischen Algorithmen enthalten, von den Studierenden implementiert, ausgetauscht und verglichen werden.

In einem nächsten Schritt könnten kryptographische Systeme aufgebaut werden, die Kommunikation von Instanzen erfordern. Dazu gehören die in Kap. 1 genannten interaktiven Kooperationen, insbesondere das Management öffentlicher Schlüssel [13, 8]. Viele der erforderlichen kryptographischen Berechnungen, die dabei lokal von jeder Instanz auszuführen sind, können mit Hilfe des CryptoManager erledigt werden.

DANK

Für wertvolle Anregungen und Diskussionen danken wir den Studenten, die im Rahmen ihres Praktikums mit dem CryptoManager gearbeitet haben. Unsere Anerkennung gilt weiterhin Birgit Pfitzmann und Andreas Pfitzmann, die diese Arbeit über viele Jahre unermüdlich und konstruktiv kritisierend begleitet haben. Nicht zuletzt danken wir Christian Eckert für seine praktische Unterstützung bei der Installation und Pflege der benötigten Softwareentwicklungsumgebung.

LITERATUR

1. L. Blum, M. Blum, M. Shub: „A Simple Unpredictable Pseudo-Random Number Generator“; *SIAM J. Comput.* 15/2 (1986) 364-383.
2. G. Bleumer: „Security for Decentralised Health Information Systems“; in B. Barber, A.R. Bakker, S. Bengtsson (ed.): *Caring for Health Information: Safety, Security and Secrecy*, Elsevier Science, Amsterdam 1994, 139-146.
3. G. Brassard: „Modern Cryptology - A Tutorial“; LNCS 325, Springer-Verlag, Berlin 1988.
4. I.B. Damgård: „Collision free hash functions and public key signature schemes“; *Eurocrypt '87*, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.
5. I.B. Damgård: „Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks“; *Crypto '91*, LNCS 576, Springer Verlag, Berlin 1992, 445-456.
6. D.W. Davies, W.L. Price: „Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer“; (2nd ed.) John Wiley & Sons, New York 1989.
7. „Specification for the Data Encryption Standard“; *Federal Information Processing Standards Publication 46* (FIPS PUB 46), January 15, 1977.
8. W. Diffie, M.E. Hellman: „New Directions in Cryptography“; *IEEE Transactions on Information Theory* 22/6 (1976) 644-654.
9. CACM: „The Digital Signature Standard Proposed by NIST“; *Communications of the ACM* 35/7 (1992) 36-40.
10. T. ElGamal: „A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms“; *IEEE Transactions on Information Theory* 31/4 (1985) 469-472.
11. D. Fox, B. Pfitzmann: „Effiziente Software-Implementierung des GMR-Signatursystems“; *Proc. Verläßliche Informationssysteme* (VIS'91), Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 329-345.
12. S. Goldwasser, Silvio Micali, Ronald L. Rivest: „A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks“; *SIAM J. Comput.* 17/2 (1988) 281-308.
13. ISO/CCITT Directory Convergence Document: „The Directory - Authentication Framework“; CCITT Recommendation X.509 and ISO 9594-8, „Information Processing Systems – Open Systems Interconnection – the Directory-Authentication Framework“.
14. X. Lai, J.L. Massey: A Proposal for a New Block Encryption Standard; *Eurocrypt '90*, LNCS 473, Springer-Verlag, Berlin 1991, 389-404.
15. B. Meyer: „Object-oriented Software Construction“; Series in Computer Science, Prentice Hall, Hemel Hempstead 1988.
16. A. Ort, R. Aßmann, G. Bleumer, M. Böttger, D. Fox, A. Pfitzmann, B. Pfitzmann, M. Waidner: „Schutz in verteilten Systemen durch Kryptologie – Ein Praktikum im Informatik-Hauptstudium“; *Datenschutz und Datensicherung* DuD 16/11 (1992) 571-579.

17. A. Pfitzmann, R. Aßmann: „Efficient Software Implementations of (Generalized) DES“; *SECURICOM 90*, March 13-16, 1990, Paris, 139-158.
18. RIPE Consortium: RIPE integrity primitives Part 1: „Final report of RACE 1040“; Centrum voor Wiskunde en Informatica, Computer Science/Departement of Algorithmics and Architecture, Report CS-R9324 , April 1993.
19. R.L. Rivest, A. Shamir, L. Adleman: „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems“; *Communications of the ACM* 21/2 (1978) 120-126,.
20. B. Schneier: „Applied Cryptography: Protocols, Algorithms, and Source Code in C“; John Wiley & Sons, New York 1994.
21. P. Zimmermann: „PGP – Pretty Good Privacy, Public Key Encryption for the Masses, User's Guide“; Volume I: Essential Topics, Volume II: Special Topics; Version 2.2 - 6 March 1993.

CryptoManager –
Eine intuitiv verwendbare Bibliothek für kryptographische Systeme

Thilo Baldin:
Institut für Informatik, Universität Hildesheim,
Samelsonplatz 1, 31141 Hildesheim, Germany,
Fax: ++49 5121 883-732

Gerrit Bleumer,
Institut für Informatik, Universität Hildesheim,
Samelsonplatz 1, 31141 Hildesheim, Germany,
Fax: ++49 5121 883-732,
e-mail: bleumer@informatik.uni-hildesheim.de

Ralf Kanne
Institut für Informatik, Universität Hildesheim,
Samelsonplatz 1, 31141 Hildesheim, Germany,
Fax: ++49 5121 883-732