Effiziente Implementierung von kryptografischen Datenaustauschformaten am Beispiel von S/MIME und OpenPGP

Matthias Schunter*
Universität des Saarlandes
Institut für Informatik
D-66123 Saarbrücken

Christian Stüble†
Universität Dortmund
Informatik LS 6
D-44221 Dortmund

Zusammenfassung

Kryptobibliotheken erlauben Programmierern, Sicherheitsfunktionalitäten in beliebige Anwendungen zu integrieren. Der Austausch von Daten zwischen verschiedenen Bibliotheken wird hierbei durch standardisierte Datenaustauschformate wie S/MIME und OpenPGP ermöglicht.

Eine globale Einigung auf ein einziges Datenaustauschformat scheint unwahrscheinlich. Daher ist es wünschenswert, daß Anwendungen durch Einbindung einer einzigen Kryptobibliothek automatisch mehrere Formate, ohne Wissen über deren Details, unterstützen können.

Dieser Artikel stellt ein von uns entwickeltes und praktisch erprobtes Verfahren zum modularen Design einer Kryptobibliothek vor. Das Ziel dieses Verfahrens ist es, mehrere Datenaustauschformate bei möglichst geringem Implementierungsaufwand (gemessen in der Gesamtlänge des Codes) zu unterstützen.

Exemplarisch demonstrieren wir das Verfahren anhand von digitalen Signaturen in OpenPGP und S/MIME v.3. Die vorgestellten Designschritte führen jedoch analog auch bei Zertifikaten oder verschlüsselten Nachrichten sowie bei anderen Datenaustauschformaten zu dem gewünschten modularen und code-minimalen Design.

1 Einleitung

Signaturen und verschlüsselte Nachrichten sind, wie alle Daten, eine Folge von Bytes ohne weitere kontextspezifische Information. Um eine Signatur zu überprüfen sind jedoch Zusatzinformationen nötig. Das sind beispielsweise Informationen über den benutzten Signaturschlüssel, die Art des benutzten kryptografischen Algorithmus und Informationen über den Signierer. Falls dies nicht fest vereinbart ist, müssen diese Informationen zusammen mit den Signaturdaten übertragen werden. Beim Austausch von

*schunter@acm.org
†stueble@ls6.cs.uni-dortmund.de

Bytefolgen zwischen verschiedenen Rechnern muß außerdem deren Interpretation definiert werden: Ein (32 Bit-) Prozessor kann die Dezimalzahl ´5´ als die hexadezimale Bytefolge ´00 00 00 05´ oder ´05 00 00 00´ kodieren. Beim Datenaustausch könnte diese somit als ´5´ oder ´83886080´ interpretiert werden. Der Kontext, nach dem eine Bytefolge ´Signatur´ oder ´Schlüsseltext´ interpretiert wird, bezeichnen wir allgemein als *Datenaustauschformat* oder kurz *Format*. Aktuelle Beispiele für solche Formate sind OpenPGP [6] und S/MIME v.3 [14].

Eine globale Einigung auf ein einziges Format scheint unwahrscheinlich. Daher ist es wünschenswert, daß Anwendungen durch Einbindung einer einzigen Kryptobibliothek automatisch mehrere Formate unterstützen können, ohne daß die Programmierer einer Anwendung hierfür Detailwissen über die Formate besitzen müssen.

Um die Entwicklung solcher Kryptobibliotheken zu vereinfachen, beschreiben wir in diesem Artikel exemplarisch einen Designprozeß einer modularen Kryptobibliothek zur Unterstützung mehrerer Formate am Beispiel von digitalen Signaturen [5, 13] und deren Formatierung mit OpenPGP und S/MIME v.3. Unser Ziel ist hierbei, mehrere Formate bei möglichst geringem Implementierungsaufwand (gemessen in der Länge des Gesamtprogrammes) in einer einzigen Kryptobibliothek zu unterstützen.

Durch eine geringe Menge von Code und einer klaren Modulstruktur mit kleinen, voneinander unabhängigen Baugruppen und klar definierten Schnittstellen sinkt sowohl der Implementierungs- und Wartungsaufwand als auch die Wahrscheinlichkeit von Fehlern in der Implementierung. Der höhere Designaufwand wird durch eine hohe Wiederverwendbarkeit des Codes (in mindestens doppelt so vielen Anwendungen) und größere Laufzeitstabilität gerechtfertigt. Desweiteren ermöglicht das hier beschriebene Verfahren das nachträgliche hinzufügen von neuen Formaten.

Abbildung 1 zeigt einen Überblick über die resultierende Struktur einer solchen Formatunterstützung: Die Anwendung kennt nur die abstrakten Funktionen “verschlüsseln” oder “signieren”. Intern werden diese Funktionen in eine formatabhängige Baugruppe “PGP Signatur” oder “S/MIME Signatur” umgesetzt. Diese Baugruppen benutzen dann die von der tieferen Schicht zur Verfügung gestellten formatunabhängigen Module, wie z.B. DSA, RSA, etc.

Eine formatunabhängige kryptographische API (*Application Programming Interface*; Abbildung 1, oberste Schicht) erleichtert dabei die Implementierung von formatunabhängigen Applikationen bzw. kryptografischen Funktionen, da vom Benutzer der Bibliothek keine formatspezifischen Besonderheiten berücksichtigt werden müssen.

Das in diesem Artikel beschriebene Designverfahren wurde beim Redesign des `CryptoManager++` [2] angewendet. Die ursprüngliche Version des `CryptoManager++` verwendete nur ein proprietäres Datenformat. Als die Bibliothek dann nachträglich um die Formate S/MIME und OpenPGP erweitert werden sollte, wurde klar, daß dieses nicht durch einige Fallunterscheidungen möglich ist. Die etwas grundsätzlichere Betrachtung der Problematik führte dann zum hier beschriebenen Analyse- und Designverfahren.

1.1 Überblick

In Kapitel 2 werden die Signaturberechnungs- und Formatierungsprozesse in ihre minimalen logischen Bestandteile, sogenannte *Module* zerlegt. Die Module, die in beiden

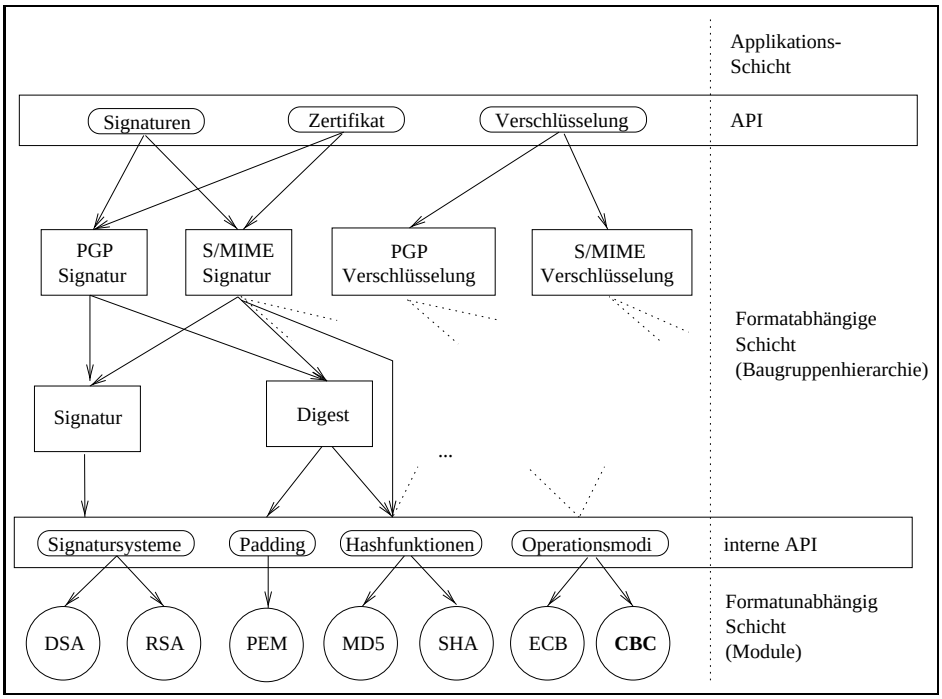

Abbildung 1: Beschreibung der zwei formatunabhängigen Schnittstellen (API) einer kryptographischen Bibliothek.

Formaten vorkommen werden anschließend in Kapitel 3 im Detail verglichen, um Kandidaten für die Zusammenfassung mehrerer Module in von beiden Formaten verwendete Einheiten zu finden. In Kapitel 4 werden diese gemeinsam verwendbaren Baugruppen beschrieben und das resultierende code-minimalen Design zusammengefaßt. Wie gewünscht spiegeln sich hier Ähnlichkeiten in den Formaten in gemeinsam verwendeten Baugruppen (und somit auch Code) wider. Eine abschließende Bewertung erfolgt schließlich in Kapitel 5.

1.2 Entwicklung von OpenPGP und S/MIME v.3

S/MIME v.3 [14] ist der vom IETF (*Internet Engineering Task Force*) spezifizierte Nachfolger von S/MIME v.2, welches auf den Formaten X.509 [17], PEM [7] und PKCS [11, 12] basiert. Der OpenPGP Standard [6] wurde aufbauend auf den Formaten PGP v2.x [9], PGP v5 und PGP/MIME [8] entwickelt, nachdem PGPv5 und auch PGPv6 von Network Associates (NAI) herausgegeben, und damit lizenzpflichtig wurde. Ziel der OpenPGP Spezifikationen ist es u.a. völlig ohne patentierte Algorithmen auszukommen. Eine ausführliche Beschreibung und ein kurzer Vergleich beider Formate findet sich in einem Artikel des *G5 Messaging Forum* [16]. Bruce Schneier stellt in [15] die Formate PEM, PGPv2 und PKCS kurz vor und beschreibt wie auch [3] alle in diesem Artikel vorkommenden kryptografischen Algorithmen.

2 Analyse der Signaturberechnungsprozesse in Module

Datentypen eines Formates sind alle die Bitfolgen, die eine semantische Bedeutung (im Zusammenhang mit dem jeweiligen Format) besitzen. Ein Beispiel sind die `SignatureSubPackages` von PGP. *Elementare Datentypen* eines Formates sind Datentypen die aus Sicht des Formates nicht zusammengesetzt sind. Beispiele sind ASN.1 Objektbezeichner in S/MIME und Algorithmenbezeichner in OpenPGP.

Als *Modul* bezeichnen wir einen aus Sicht der Formatierung unteilbaren Schritt des Formatierungsprozesses welcher ausschließlich auf elementaren Datentypen arbeitet. Beispiele sind die Berechnung einer Hashfunktion oder einer Signatur.

Eine *Baugruppen* ist eine Zusammenfassung mehrerer Module in eine funktionale Einheit. Eine Baugruppe kann beliebige Datenstrukturen als Ein- und Ausgabe verwenden. Ein Beispiel ist die komplette Berechnung und Formatierung einer OpenPGP Signatur.

Basierend auf den Signaturdatenstrukturen von OpenPGP und S/MIME v.3 wird nun der Signaturprozeß in seine modularen Bestandteile zerlegt, indem in einem top-down Verfahren Baugruppen so lange rekursiv zerlegt werden, bis die gesamte Signaturerzeugung in Module, die auf elementaren Datentypen arbeiten, zerlegt ist.

2.1 Zerlegung des OpenPGP Signaturprozesses

Abbildung 2 zeigt die Datenstruktur einer OpenPGP Signatur (Signaturversion 3), die als `SignaturePacket` bezeichnet wird[1]. Der Erzeugungsprozeß muß neben einem OpenPGP spezifischen Header (mit Längenbeschreibung und Versionsnummer) und der eigentlichen Signatur weitere Daten einfügen: Hinter dem Versionsbyte befinden sich die `SignatureSubPackets`, welche mit der Nachricht mitsigniert werden. Bis zur Signaturpacketversion 3 bestanden die Subpackets aus einem Byte, welches die Art der Signatur genauer spezifiziert hat und ein 32-Bit Timestamp, dem Zeitpunkt der Signaturberechnung. In neueren Signaturversionen sind viele neue `SubPacket`-Typen hinzugekommen, von denen einige vom Anwender bestimmt werden können. Die Gesamlänge aller `SignatureSubPackets` muß von einem Subprozeß berechnet und als Byte vor dem ersten Subpacket eingefügt werden. Zur Identifikation des öffentlichen Schlüssel des Signierers dient die aus diesem berechnete Key-ID. Danach folgt ein Bezeichner der verwendeten Hashfunktion. Um testen zu können, ob die richtige Hashfunktion zur Berechnung des Digest benutzt wurde, muß ein weiterer Subprozeß zwei Kontrollbytes des Digest im `SignaturePacket` einfügen.

Die Signatur wird von einem Signierprozeß aus einer Datenstruktur `DigestInfo` (siehe Abb. 5) erzeugt. Diese ASN.1 Struktur ist zusammengesetzt aus einem ASN-Bezeichner des verwendeten Hashalgorithmus und dem berechneten Digest. Der Digest wird durch eine Hashfunktion, welche vom Benutzer gewählt werden kann, aus den zu signierenden Daten berechnet. Diese werden wiederum aus der zu signierende Nachricht durch Anhängen aller `SignatureSubPackets` erzeugt.

[1] Alle OpenPGP Nachrichten sind nach Zweck in sogenannte Pakete gegliedert wobei der Pakettyp im ersten Byte des Pakets angegeben wird. Weitere Pakettypen sind u.a. `PublicKeyEncryptedPacket` und `PlaintextPackage`.

Anz. Bytes	*Inhalt*
1	Packet Identifier
2	8-bit or 16-bit length of this packet
1	Version byte (=3)
1	Length of following material that is included in calculation of digest
1	Signature classification field (SCF)
1	32-bit time/date of signing
8	64-bit key ID
1	Algorithm Identifier for public key scheme
1	Algorithm Identifier for message digest
2	First two bytes of message digest
(keylen)	MPI encrypted message digest

Abbildung 2: Struktur eines PGP Signature-Packets.

```
SignedData ::= SEQUENCE {
  version          Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  encapContentInfo EncapsulatedContentInfo,
  certificates     [0] IMPLICIT CertificateSet OPTIONAL,
  crls             [1] IMPLICIT CertificateRevocationLists
                       OPTIONAL,
  signerInfos      SignerInfos}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

Abbildung 3: Struktur einer S/MIME Signatur ASN.1 Typ `SignedData` (siehe Abb. 4 für den ASN.1 Typ `SignerInfo`).

Ein Überblick über diesen in Module zerlegten Prozeß zusammen mit den stattfindenden Datenflüssen enthält Abbildung 6.

2.2 Zerlegung des S/MIME Signaturprozesses

Eine S/MIME Signatur ist ein ASN Datentyp `SignedData`, der in Abbildung 3 wiedergegeben ist. Eine signierte Nachricht `SignedData` kann, im Gegensatz zu PGP, von mehreren Signierern signiert sein. Die einzelnen Signaturen `SignerInfo` sind in Abb. 4 wiedergegeben. Desweiteren enthält die signierte Nachricht eine Liste `digestAlgorithms` mit allen verwendeten Hashfunktionen, eine Beschreibung `encapContentInfo` der signierten Nachricht, eine optionale Liste `certificates` von Zertifikaten und eine optionale Liste `crls` mit Certificate Revocation-Lists.

Die ASN.1 Datenstruktur `SignerInfo` besitzt neben der Signatur und den Bezeichnern der verwendeten Algorithmen unsignierte (`unsignedAttrs`) und signierte (`signedAttrs`) Attribute und einen eindeutigen Bezeichner `issuerAndSerialNumber` des

```
SignerInfo ::= SEQUENCE {
 version               Version,
 issuerAndSerialNumber IssuerAndSerialNumber,
 digestAlgorithm       DigestAlgorithmIdentifier,
 signedAttrs           [0] IMPLICIT SignedAttributes
                           OPTIONAL,
 signatureAlgorithm    SignatureAlgorithmIdentifier,
 signature             SignatureValue,
 unsignedAttrs         [1] IMPLICIT UnsignedAttributes
                           OPTIONAL}

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute
1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
 attrType OBJECT IDENTIFIER,
 attrValue SET OF AttributeValue}

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

Abbildung 4: Aufbau des ASN.1 Typs SignerInfo (Teil der S/MIME Signatur in Abb. 3).

```
DigestInfo ::= SEQUENCE {
 digestAlgorithm       DigestAlgorithmIdentifier,
 digest                Digest
}

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

Digest ::= Bytestring
```

Abbildung 5: Der Aufbau des ASN.1 Typs DigestInfo, der sowohl in OpenPGP als auch in S/MIME als Eingabetyp für das Padding dient.

Signierers. Vor der Berechnung der Signatur werden alle zu signierenden Attribute an die Nachricht angehängt und diese dann in einen ASN.1 Datentyp konvertiert. Weitere Informationen zu der Nachricht werden im Feld encapContentInfo eingetragen. Aus der Nachricht wird dann mittels einer Hashfunktion der Digest berechnet, der dann in den ASN.1 Datentyp DigestInfo (siehe Abb. 5) gewandelt wird. Der Algorithmenbezeichner der verwendeten Hashfunktion muß sowohl in der Datenstruktur SignerInfo, als auch in den Datenstrukturen SignedData und DigestInfo eingetragen werden. Nachdem die DigestInfo-Struktur kompatibel zu PEM/PKCS gepaddet wurde, kann

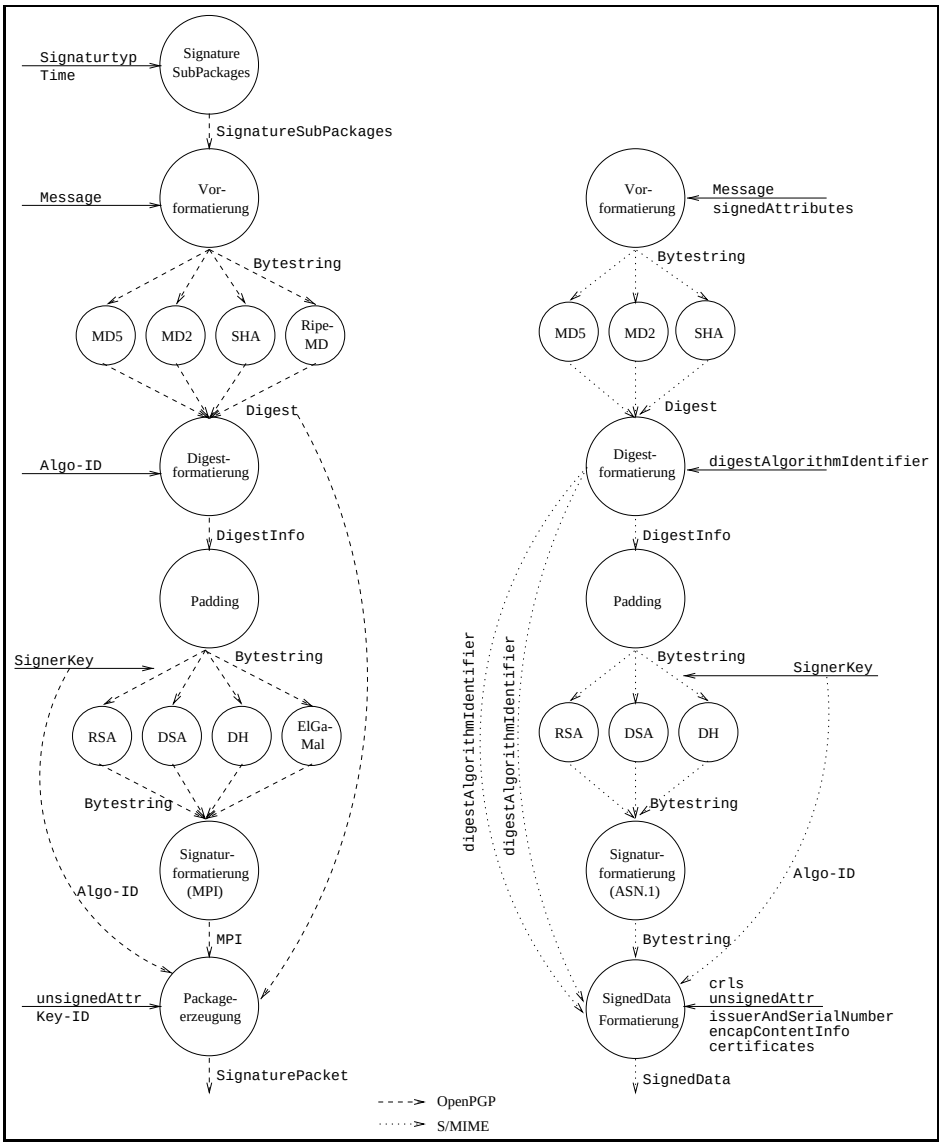

Abbildung 6: Beschreibung der Abhängigkeiten und des Datenflusses zwischen den für OpenPGP und S/MIME Signaturen benötigten Modulen (Gestrichelte Pfeile im linken Teil bezeichnen die Datenflüsse von OpenPGP, gepunktete im rechten Teil die von S/MIME).

mit dem Signaturverfahren die Signatur berechnet werden. Diese wird dann in den ASN.1 Datentyp OCTET STRING gewandelt.

Ein Überblick über diesen in Module zerlegten Prozeß zusammen mit den stattfindenden Datenflüssen enthält Abbildung 6.

3 Vergleich der Module beider Prozesse

In diesem Kapitel werden zuerst alle Module, die in beiden Formaten ähnlich scheinen, verglichen. Gemeinsam verwendete Module werden dann in Kapitel 4 zu formatunabhängigen Baugruppen zusammengefaßt. Eine Baugruppe umfaßt hierbei alle benachbarten formatunabhängigen Module.

Vorformatierung: Die Vorformatierung der zu signierenden Bytefolge, bestehend aus Nachricht und zu signierenden Attributen, muß formatabhängig geschehen, weil die verwendeten Datentypen der Attribute und der Nachricht unterschiedlich sind. Der verwendete Signaturschlüssel wird in beiden Formaten auf unterschiedliche Art und Weise re-identifiziert, so daß auch diese Module getrennt implementiert werden müssen. Da sich auch die Datentypen der Algorithmenidentifikatoren unterscheiden, müssen die Module, die diese Informationen generieren, auch getrennt implementiert werden.

Digestberechnung: Die Hashfunktionen, soweit beide Formate sie unterstützen, können doppelt benutzt werden, da sie letztendlich auf Bytefolgen arbeiten.

Digestformatierung: Beide Formate konvertieren den berechneten Digest in den ASN.1 Typ `DigestInfo` und verwenden in dieser Struktur sogar kompatible Algorithmen-bezeichner, so daß auch diese Module formatunabhängig benutzt werden können.

Padding: Padding dient dem Auffüllen von Daten variabler Länge bis zu einem Vielfachen der vorgegebene Blockgröße. Das Modul zum Padding der Eingabedaten kann gemeinsam benutzt werden, da beide Formate das in PEM/PKCS beschriebene Paddingverfahren unterstützen.

Signaturberechnung: Die Signaturfunktion selbst arbeitet wiederum auf einer Bytefolge, so daß diese auch nur einmal implementiert werden muß.

Signaturformatierung: Die erhaltene Signatur wird bei S/MIME in ein ASN.1 Datentyp und bei OpenPGP in eine MPI-Zahl konvertiert. Diese Module müssen also wieder getrennt implementiert werden.

4 Zusammenfassung gleicher Module in formatunabhängige Baugruppen maximaler Größe

4.1 Baugruppe `Digest`

Da sowohl die Berechnung der Hashfunktionen als auch die weitere Konvertierung des Digest in den ASN.1 Datentyp `DigestInfo` formatunabhängig ist, können beide Module zu einer Baugruppe `Digest` (Abbildung 7) zusammengefaßt werden.

4.2 Baugruppe `Signatur`

Die Module `Padding` und `Signaturberechnung` können zu einer weiteren Baugruppe `Signatur` (Abbildung 8) zusammengefaßt werden[2].

[2]Obwohl beide Baugruppen formatunabhängig sind, kann die Baugruppe `Digest` nicht ohne Nebenbedingungen mit der Baugruppe `Signatur` zusammengelegt werden, da zur Generierung eines OpenPGP `SignaturePackets` ein Zugriff auf den Digest nötig ist. Es wäre jedoch eine Baugruppe `HybridSign` denkbar,

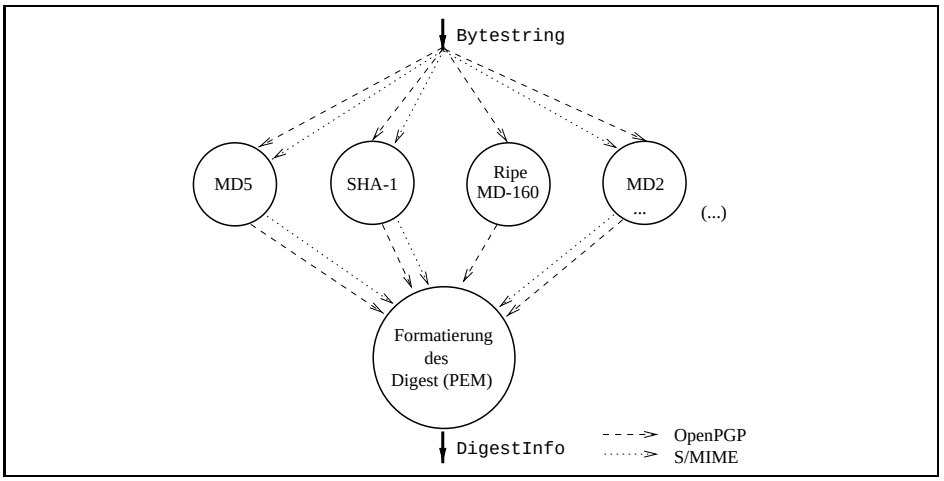

Abbildung 7: Zusammenfassung der Hashfunktionen und der Digestformatierung zu einer formatunabhängigen Baugruppe `Digest`.

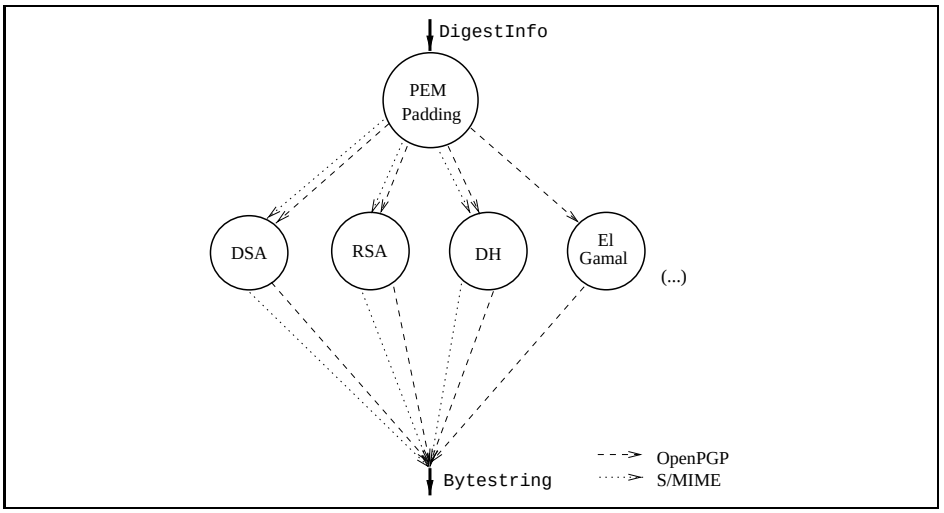

Abbildung 8: Zusammenfassung des Paddingmoduls und der Signaturverfahren zu einer formatunabhängigen Baugruppe `Signatur`.

Die Ausgabe dieser Baugruppe kann dann von einem formatinternen Modul weiterverarbeitet werden. Alle anderen Module und Baugruppen des Signaturberechnungsprozesses sind zu formatspezifisch, um sie in formatunabhängigen Baugruppen zusammenfassen zu können. Auch die abschließende Formatierung der OpenPGP oder S/MIME Signaturstruktur aus den Ergebnissen der anderen Module und Baugruppen ist zu formatspezifisch, um formatübergreifend implementiert zu werden.

---

die als Rückgabewert den Datentyp `DigestInfo` und die berechnete Signatur liefert.

4.3 Ergebnis

Abbildung 9 zeigt, welche Module und Baugruppen aufgrund der vorangegangenen Analyse zur Erzeugung einer OpenPGP- oder S/MIME Signatur nötig sind. Von den vielen benötigten Funktionen sind nur die Baugruppen `Digest` und `Signatur` gemeinsam benutzbar. Daher scheint es sinnvoll zu sein, die Erzeugung der Signatur für jedes Format getrennt zu implementieren, beide Implementierungen jedoch gemeinsam die formatunabängigen Baugruppen `Digest` und `Signatur` benutzen zu lassen. In einem höheren Abstraktionslevel können die formatabhängigen Implementierungen dann wieder als eine Baugruppe mit gleicher Schnittstelle zusammengefaßt werden. Diesen Sachverhalt stellt Abbildung 1 dar, indem der Anwendung eine abstrakte Schnittstelle “Signaturen” zur Verfügung gestellt wird, die von formatspezifischen Baugruppen implementiert wird, welche wiederum auf formatunabhängige Module und Bauguppen zurückgreifen.

4.4 Objekt-orientierte Implementierung

Das in diesem Artikel beschriebene Analyse- und Designverfahren wurde im Rahmen eines Redesigns des `CryptoManager++` [2] entwickelt und erprobt. Im derzeitigen Prototyp sind die formatunabhängigen Module und Baugruppen in einem sogenannten Kernel implementiert (Vgl. `www-krypt.cs.uni-sb.de/~cm`). Die Zusammenfassung und die formatspezifischen Module sind hingegen Teil einer darüberliegenden formatspezifischen Schicht, die z.B. die Klassen `PGPEncrypter` oder `SMIMEEncrypter` beinhaltet. Da in dieser Schicht die formatspezifische Klassen die gleiche Schnittstelle besitzen, kann eine Applikation beispielsweise formatunabhängig OpenPGP- oder S/MIME-Verschlüsselungen vornehmen.

5 Fazit und Ausblick

In diesem Beitrag wurden die kryptografische Datenaustauschformate OpenPGP und S/MIME analysiert, die benötigten Prozesse zu deren Formatierung in Module unterteilt (Abbildung 6) und gleiche Module dann zu formatunabhängigen Baugruppen kombiniert. Das resultierende code-minimale Design ist in Abbildung 9 zusammengefaßt. Es spiegelt die Gemeinsamkeiten der Signaturformate von S/MIME und OpenPGP optimal in gemeinsamem Code wider.

Die gemeinsame Implementierung von OpenPGP und S/MIME Signaturen benötigt nur noch 17 Module im Vergleich zu 25 Modulen für eine getrennte Implementierung (14 Module für OpenPGP und 11 Module für S/MIME). Die Einsparung von Code ist jedoch mehr als 30%, da die gemeinsamen Baugruppen alle aufwendigen kryptografischen Algorithmen umfassen.

Das Hinzufügen weiterer Signaturformate kann nun im nachhinein nach dem gleichen Verfahren erfolgen: Nach einer Analyse des neuen Formats in Module wird für jedes Modul geprüft, ob diese schon als Module oder Baugruppen eines anderen Formats existiert. Falls dies der Fall ist, kann die bestehende Implementierung verwendet werden. Hierbei steigt die Wahrscheinlichkeit, daß benötigte Module bereits imple-

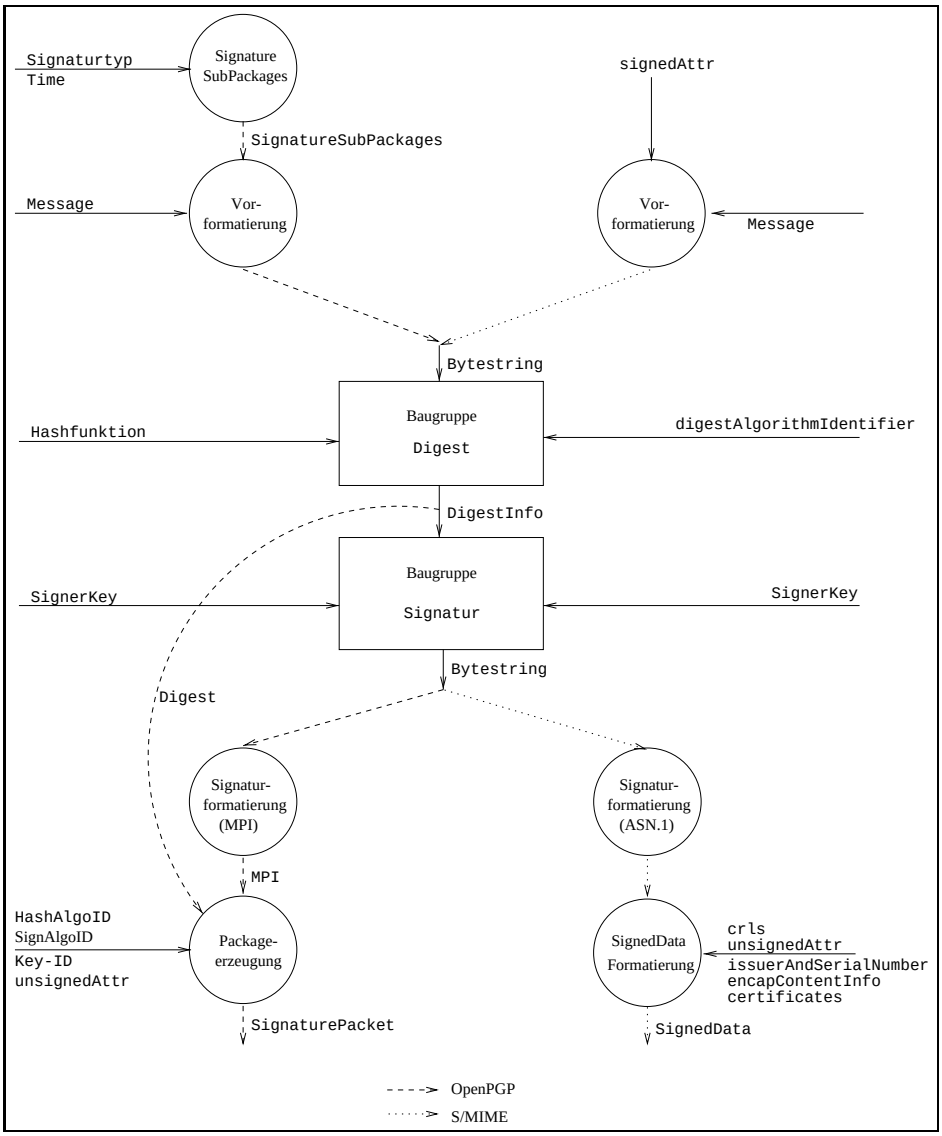

Abbildung 9: Beschreibung der benötigten formatabhängigen Module bei Verwendung der formatunabhängigen Baugruppen Digest und Signatur.

mentiert wurden, bei zunehmender Anzahl von unterstützten Formaten. Falls es das Modul noch nicht gibt, muß die Bibliothek entsprechend ergänzt werden. Dies wird auch an unserem Beispiel ersichtlich: Das nachträgliche Hinzufügen von S/MIME zu einer nach unserem Verfahren analysierten und implementierten Bibliothek für OpenPGP benötigt nur noch 3 neue Module, was eine erhebliche Einsparung im Vergleich zu den 11 zusätzlich nötigen Modulen bei einer getrennten Implementierung darstellt.

Neben den Signaturformaten kann das vorgestellte Verfahren auch auf Verschlüsselungs- und Zertifikatmanagementformate angewendet werden, wobei sich auch grössere Einsparungen als in unserem Beispiel ergeben könnten.

Als eine besondere Schwierigkeit bei der Entwicklung von Baugruppen zur Verschlüsselung könnte sich herausstellen, daß OpenPGP einen etwas abgewandelten CFB-Operationsmode [9] für die symmetrische Verschlüsselung benutzt. Bei den Zertifikatmanagementfunktionen müssen die verschiedenen Vertrauensmodelle und die damit verbundenen Unterschiede in den Zertifikatstrukturen berücksichtigt werden.

6 Danksagung

Wir danken Tom Beiler, Ralph Kühl, Birgit Pfitzmann und Sven Wohlgemuth für hilfreiche Ratschläge zur Verbesserung des Artikels. Die Idee mehrere Formate mit einer Bibliothek zu unterstützen stammt von Gerrit Bleumer. Matthias Schunter wurde teilweise vom Projekt ACTS.SEMPER <www.semper.org> unterstützt.

Literatur

[1] Specification of Abstract Syntax Notation One (ASN.1); CCITT Recommendation X.208 (1988).

[2] Thilo Baldin, Gerrit Bleumer; CryptoManager++ - an object oriented software library for cryptographic mechanisms; 12th IFIP International Conference on Information Security (IFIP/Sec´96), Chapman & Hall, London 1996, 489-491.

[3] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone; Handbook of Applied Cryptography; CRC Press, Boca Raton 1997.

[4] Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER); ISO/IEC 8825-1;

[5] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM Journal on Computing 17/2 (1988) 281-308.

[6] OpenPGP Message Formats; Internet Draft draft-ietf-openpgp-formats-*.txt

[7] D. Balenson; Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes and Identifiers; RFC 1423, TIS, Oktober 1993

[8] MIME Security with Pretty Good Privacy (PGP); RFC 2015, October 1996

[9] Atkins, Stallings, P. Zimmermann; PGP Message Exchange Formats; RFC 1991, August 1996

[10] PGP International Home Page http://www.pgpinternational.com/

[11] RSA Laboratories; PKCS 1: RSA Encryption, Version 2; http://www.rsa.com/rsalabs/pubs/PKCS/html/pkcs-1.html

[12] RSA Laboratories; PKCS 6: Extendend-Certificate Syntax Standard, Version 1.5; November 1998.

[13] 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, reprinted: 26/1 (1983) 96-99.

[14] Cryptographic Message Syntax; Internet Draft draft-ietf-smime-cms-*.txt

[15] Bruce Schneier; Applied Cryptography; John Wiley & Sons, Inc; 1996

[16] http://www.group5forum.org/

[17] The Directory Authentication Framework; CCITT Recommendation X.509 (1988).