

# Anwenderhandbuch Packer

V4.4.3 / Stand: 11.01.2024

Erstellt im Auftrag des Gemeinsamen  
Bundesausschusses

# Inhaltsverzeichnis

Impressum .....	3
Informationen zu diesem Dokument .....	4
1 Einleitung .....	9
2 Voraussetzungen .....	10
3 Hintergrund der Verschlüsselung .....	11
3.1 Schlüsselformate .....	12
4 Datei-Formate und Prozesse .....	13
5 Allgemeines zu Java-Programmen .....	14
6 TPacker .....	15
7 XPacker .....	20
Basisfunktionen .....	20
Ergänzende Funktionen .....	22
8 GPacker .....	24
8.1 Schlüsselverwaltung .....	24
8.2 P-Mode .....	31
8.3 K-Mode .....	38
8.4 Fehlermeldungen .....	41
9 Anwendungsbeispiele .....	43
9.1 Leistungserbringer (Kliniken/Praxen) .....	43
9.2 Datenannahmestellen .....	44
10 Technische Spezifikation .....	46
10.1 Hintergrund der K-Mode-Verschlüsselung .....	46
10.2 Spezifikation der K-Mode-Verschlüsselung .....	46

# Impressum

**Thema:**

Anwenderhandbuch Packer V4.4.3

**Auftraggeber:**

Gemeinsamer Bundesausschuss

**Datum der Abgabe:**

11.01.2024

**Herausgeber:**

IQTIG – Institut für Qualitätssicherung und Transparenz im Gesundheitswesen

Katharina-Heinroth-Ufer 1  
10787 Berlin

Telefon: (030) 58 58 26-0  
Telefax: (030) 58 58 26-999

[info@iqtig.org](mailto:info@iqtig.org)  
<http://www.iqtig.org>

# Informationen zu diesem Dokument

## Darstellungsmittel

Im Folgenden sind Symbole und Darstellung besonderer Informationen beschrieben.



### Achtung

Beschreibt Ursache, Folge und Vermeidung einer besonderen Fehlanwendung, die zu Problemen bei der Implementierung oder Ähnlichem führen kann.

## Zielgruppe

Dieses Handbuch richtet sich an administrative Mitarbeiter sämtlicher am Datenfluss beteiligter Stellen und Softwareentwickler, die mit der Umsetzung der XML-Datenflüsse und der damit verbundenen Verschlüsselung einzelner XML-Elemente sowie vollständiger XML-Dateien beschäftigt sind.

## Änderungen in der Version 4.4.3

- Anpassung der Dokumentation an das Corporate Design
- Erweiterung des XPackers um folgende Funktionen:
  - Ausgabe des Modulus eines zu übergebenden öffentlichen Schlüssels
  - Ausgabe eines Wahrheitswertes, der angibt, ob ein bestimmtes XML-Tag einer XML-Datei mit einem bestimmten öffentlichen Schlüssel verschlüsselt worden ist.
  - Die beiden Funktionen stehen auch über die Java-API des XPackers zur Verfügung

## Änderungen in der Version 4.4.2

- Einführung des Profils "Datenflussprotokoll Strukturabfragen" im P-Mode des GPackers zur Transportverschlüsselung ohne Registriernummer
- Anpassung von Schriftart und Icon an das Corporate Design auf der Benutzeroberfläche des GPackers
- Technische Spezifikation der K-Mode-Verschlüsselung im Benutzerhandbuch (siehe Kapitel 10)
- Ersetzen der MagicBytes "RPackerv1" durch "KMODEv001" bei der K-Mode-Verschlüsselung
- Verbesserung der Definition der Zufallsquelle in Klasse "CryptographySupportShared".

## Änderungen in der Version 4.4.1

- Ermöglichung des Einlesens von mehrzeiligen Schlüsseldateien mit Unix (LF)-Zeilenende
- Zulassen von Binde- und Unterstrich als Bestandteil des Schlüsselnamens
- Erhöhung der log4j-Version von 2.17.1 auf 2.20.0, Erhöhung der slf4j-Version von 1.7.25 auf 2.0.7 und Austausch von Abhängigkeit org.apache.logging.log4j:log4j-slf4j-impl:2.17.1

gegen Abhängigkeit `org.apache.logging.log4j:log4j-slf4j2-impl:2.20.0`.

### Änderungen in der Version 4.4.0

Hinweis: Version wurde nicht veröffentlicht.

- Erweiterung des GPackers um Schlüsselverwaltung und K-Mode
- GParser verarbeitet jetzt auch Dateien mit groß geschriebener Dateieindung wie z.B. "datei.ZIP.AES"
- Verschlüsselung von Zertifikatsdateien (.cer) im P-Mode und K-Mode
- Einführung der Rolle "Klinisches Krebsregister"
- Umzug von TParser-Core-Crypto-Klassen in Parser packer-shared-Modul. Diese werden nun gemeinsam vom TParser und GParser genutzt. Dabei wurden Klassen vom Paket `org.iqtig.tparser.core.impl.rparser` nach `org.iqtig.packer.shared.crypto` bzw. nach `org.iqtig.packer.shared.error.crypto` verschoben.
- Erweiterung der TParser-API und der XParser-API um die Möglichkeit, entsprechende Methoden mit einem Public-Key oder Private-Key-Objekt aufzurufen. Bis dahin war nur der Aufruf mit Schlüsseldateipfad möglich. Außerdem kann bei Nutzung der neuen Methoden auch eine Key-Id übergeben werden als Bestandteil der Schlüssel-Id, die im KeyInfo-Block des XMLs als Id-Attribut eingesetzt wird.
- Umstieg der Dokumentation auf Ascii-doc
- Ergänzung der Dokumentation um neue Funktionen des GPackers

### Änderungen in der Version 4.3.1

- TParser-Core-jar zu Release-Bündel hinzugefügt
- Abhängigkeit `org.apache.santuario:xmlsec:2.1.4` des XPackers auf Version 2.3.0 angehoben

### Änderungen in der Version 4.3.0

Hinweis: Im Release-Bündel dieser Version fehlt das TParser-Core-jar.

- Parser benötigen jetzt mindestens Java 11
- TParser
  - Umstellung des Kommandozeilen-Interfaces auf picocli. Dadurch leicht veränderte Syntax der Befehlsaufrufe. Siehe Bemerkung im Kapitel 6 TParser.
  - Ausstattung des XPackers mit der Funktion der K-Mode-Verschlüsselung (hybride Verschlüsselung).
  - Ausstattung des XPackers mit einer Funktion, die prüft, ob eine Datei im K-Mode verschlüsselt worden ist.
  - Möglichkeit beim TParser bezüglich der K-Mode-Entschlüsselung, den privaten Schlüssel beim Entschlüsseln aus einem PKCS12-Store zu lesen.
  - Bugfix: Versuchte man eine Datei zu Entschlüsseln und Entpacken und gab man ein Output-Verzeichnis, das nicht existierte an, wurde die falsche Fehlermeldung

ausgegeben, dass das Passwort falsch sei. Jetzt wird ein entsprechender Fehler ausgegeben, dass das Ausgabeverzeichnis nicht existiert.

- Änderungen am Code/Konfiguration:
  - Manuelle close-Anweisungen minimiert.
  - Logging auf slf4j/log4j2 umgestellt.
  - TPACKer in Module Tpacker-Cli und TPACKer-Core aufgeteilt.
- XPACKer
  - Bei der internen Verarbeitung zu komprimierender und zu verschlüsselnder Elemente wie z.B. „qs\_data“ wird vor dem komprimierten Element ein Kommentar "<!-- Komprimiert mit XPACKer >= v0.4 -->" hinzugefügt. Beim Entschlüsseln und Dekomprimieren wurde bisher ausschließlich genau dieser Kommentar vor dem Dekomprimieren entfernt. Neu ist, dass jetzt an dieser Stelle vor dem Dekomprimieren beliebige Kommentare wie „<!--GZip-->“ entfernt werden. So kann der XPACKer auch Dateien Entschlüsseln und Entpacken, die von anderen Anwendungen/Bibliotheken gepackt und verschlüsselt worden sind.
  - In dem KeyInfo-Element des XMLs werden jetzt auch die öffentlichen Schlüssel abgelegt, die bei der XML-Verschlüsselung zum Einsatz kamen.

#### Änderungen in der Version 4.2.14

- Packer kann wieder mit Java ab Version 8 betrieben werden.

#### Änderungen in der Version 4.2.13

- Upgrade der Log4j Dependency Version auf 2.17.0

#### Änderungen in der Version 4.2.12

- Upgrade der Log4j Dependency Version auf 2.16.0
- Upgrade der BouncyCastle Dependency Version auf 1.70
- Bugfix: TPACKer: Eine Entschlüsselung war nicht möglich, wenn das Passwort Sonderzeichen enthielt und die Verschlüsselungsumgebung (JVM auf dem Betriebssystem) eine andere Standardzeichencodierung als die Entschlüsselungsumgebung nutzte. Grund dafür war, dass der TPACKer bis dato immer das Passwort mit der Zeichencodierung der Umgebung interpretierte (JVM-file.encoding).
  - **Novum: Verschlüsselung:** Das Passwort darf fortan nur Zeichen aus folgender Zeichenmenge beinhalten: a-zA-Z0-9.,;:=+\*#\_!\$%?@(){}|^~
  - **Novum: Allgemein:** Der TPACKer interpretiert das Passwort immer einheitlich und plattformunabhängig: Ausnahme: siehe nächster Punkt.
  - **Novum: Entschlüsselung:** Es existieren folgende Lösungswege, sofern mit dem TPACKer Altdaten entschlüsselt werden müssen, die mit einer früheren Version des TPACKers, mit einem Sonderzeichenpasswort, auf einer Plattform mit abweichendem Standard-Encoding, verschlüsselt wurden:
    - Konsolenanwendung: Für die selten abzusehenden Problemfälle wurde für die

Entschlüsselung der optionale Parameter `-c` (`--characterset_pw_decrypt`) eingeführt, mit dem die Zeichencodierung des Passworts zur Entschlüsselung festgelegt werden kann.

- API: Der entsprechende API-Parameter (zu `-c` auf der Konsole) ist dokumentiert unter: `org.iqtig.tpacker.Api`
  - Sofern ausschließlich mit dem GPacker entpackt wird, kann für die Problemfälle die bisherige Version 4.2.10 behalten werden. Sofern das Problem erstmalig auftritt, muss dieser mit `-Dfile.encoding=<Zeichencodierung wie auf der verschlüsselnden Plattform>` gestartet werden.
- Anpassungen in diesem Dokument:
    - Absatz zum TPacker angepasst: Neuen optionalen Parameter `-c` samt Beispielen aufgeführt.

### Änderungen in der Version 4.2.10

- Absatz zu Schlüsselformaten hinzugefügt

### Änderungen in der Version 4.2.9

- Anmerkung zum Umgang mit Speicherproblemen hinzugefügt.

### Änderungen in der Version 4.2.8

- Die Version dieses Handbuchs wird fortan mit dem Versionsstand der Software angegeben, d.h. Version 4.2.8 statt – wie zu erwarten – Version 6.
- API: Library-Updates (gemäß OWASP-Empfehlung): `santuario.xmlsec 2.1.3` → `2.1.4`
- Bugfix: TPacker/XPacker: Parameter konnten nicht mehr zusammengerückt werden (bspw. nicht `-ze` statt `-z -e`, oder nicht `-zeyt` statt `-z -e -y --timestamp`)

### Änderungen in der Version 05

- Bugfix: Die Ausgabe der Hilfe (`-h`) der Konsolenanwendungen funktionierte nicht.
- Neue Profile wurden dem GPacker hinzugefügt:
  - LE - Leistungserbringer (Datenlieferung mit Transplantationen)
  - LE - Leistungserbringer (Datenlieferung ohne Transplantationen)
- API
  - Die `pom.xml`-Dateien, die den API-JAR-Dateien beigelegt sind, waren veraltet und unvollständig.
    - Die `pom.xml`-Dateien führen jetzt alle aktuellen, transitiven Abhängigkeiten der API-JARs auf.
    - Öffentlich verfügbare Abhängigkeiten (bspw. `org.bouncycastle:bcprov-jdk15on`) werden nicht gesondert mit ausgeliefert.
  - Irrelevante Attribute wurden aus `ValidationErrorException` entfernt, bspw. `EnDataStatus:status`.

- Package-Verschiebungen/Umbenennungen
  - `org.iqtig.packer.util` > `org.iqtig.packer.shared`
  - `org.iqtig.packer.util.error.ErrorsFromDb.Dechiffrierung` > `org.iqtig.packer.shared.error.Errors.*`
  - `org.iqtig.packer.util.error.ErrorsFromDb.Verschlueselungsprogramm` > `org.iqtig.packer.shared.error.Errors.*`

#### **Änderungen in der Version 04**

- Grundsätzliche Überarbeitung nach Sanierung des GPackers:
  - zur Unterstützung des XML-Elements *patient\_tx*
  - nach Entfernung des XML-Element *feedback\_key*
- Entfernung des Kapitels Aktionen
- Übernahme von Informationen aus Textdatei in Kapitel 7 XPacker
- Überarbeitung der Erläuterungen in Kapitel 8 GPacker
- Aktualisierung von Kapitel 9 Anwendungsbeispiele

#### **Änderungen in der Version 03**

- Anpassung der Systemvoraussetzungen in Kapitel 2

#### **Änderungen in der Version 02**

- Anpassung der Systemvoraussetzungen in Kapitel 2 (JAVA 8)
- Einführung einer Versionierung dieses Anwenderhandbuchs
- Aufnahme des Abschnitts „Informationen zu diesem Dokument“

# 1 Einleitung

Dieses Anwenderhandbuch dokumentiert die Packer-Anwendungen des IQTIGs. Sie dienen zur Transformation von Dateien im XML-Format zum Austausch im Rahmen von DeQS-RL <sup>[1]</sup> oder oKFE-RL <sup>[2]</sup>.

Die Anwendungen sind:

1. TPacker – Konsolenanwendung zur Transportverschlüsselung
2. XPacker – Konsolenanwendung zur XML-Inhaltsverschlüsselung
3. GPacker – Grafische Anwendung zur Nutzung von TPacker und XPacker

[1] Richtlinie zur datengestützten einrichtungsübergreifenden Qualitätssicherung

[2] Richtlinie für organisierte Krebsfrüherkennungsprogramme

## 2 Voraussetzungen

Die Programme setzen die Installation von Java 11 oder einer höheren Version voraus. Dies gilt insbesondere für die Nutzung von OpenJDK. Ferner werden – je nach Aktion – „private“ oder „öffentliche Schlüssel“ benötigt. Diese werden im folgenden Abschnitt erklärt.

### 3 Hintergrund der Verschlüsselung

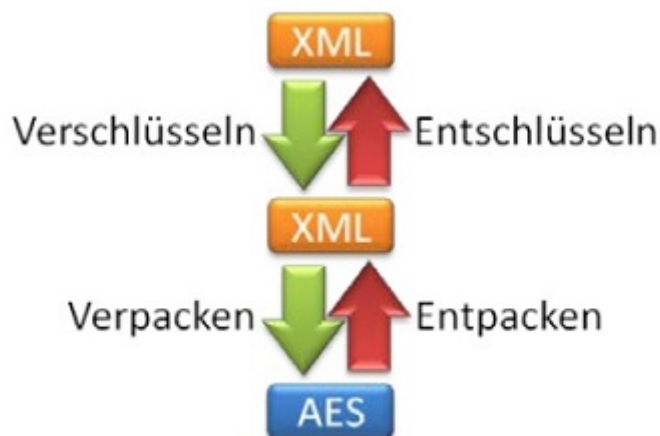
Für die XML Ver- und Entschlüsselung werden sogenannte Schlüssel benötigt. Diese Schlüssel bestehen aus langen Zeichenketten und werden daher in Dateien abgelegt. Da nun mehrere Organisationen diese Dateien, aber nicht jede Organisation alle Daten darin sehen/lesen darf, werden Schlüsselpaare eingesetzt. Jedes Schlüsselpaar besteht aus zwei unterschiedlichen Typen von Schlüsseln:

- Private Schlüssel: Der eigene private Schlüssel darf niemals dem Kommunikationspartner gegeben werden.
- Öffentlicher Schlüssel: Nur der öffentliche Schlüssel darf dem Kommunikationspartner gegeben werden.

Wenn Daten des Absenders A für verschiedene Empfänger (E1, E2, usw.) verschlüsselt werden sollen, so benötigt man die öffentlichen Schlüssel von E1 und E2 zum Verschlüsseln der Daten von A: Der Bereich der nur für E1 ist, wird mit dessen öffentlichem Schlüssel chiffriert. Analog mit dem Bereich für E2 usw.

Wenn E1 die Daten erhält, kann E1 mit seinem eigenen privaten Schlüssel die für ihn bestimmten Daten entschlüsseln. Er kann nicht die für E2 bestimmten Daten entschlüsseln. Dieses Verfahren wird auch in der Qualitätssicherung eingesetzt. Die Leistungserbringer verschlüsseln Daten mit verschiedenen öffentlichen Schlüsseln der Empfänger, welche dann nur die für sie bestimmten Daten entschlüsseln und verarbeiten können.

Abbildung 1: Ver- und Entschlüsselung sowie Ver- und Entpacken der XML-Elemente



#### Wichtige Hinweise

- Beim Verschlüsseln nur den öffentlichen Schlüssel des Empfängers einsetzen
- Beim Entschlüsseln nur den eigenen privaten Schlüssel einsetzen
- **Niemals den eigenen privaten Schlüssel an Dritte weitergeben!**
- Nur öffentliche Schlüssel an Dritte geben

- Die Schlüssel werden folgendermaßen bezeichnet:

Privater Schlüssel:	' Englisch: Private Key '	Dateiendung: *.pri
Öffentlicher Schlüssel:	' Englisch: Public Key '	Dateiendung: *.pub

*Die Bereiche, die unterschiedlich verschlüsselt werden müssen, werden auch als "Elemente" in der Anwendung bezeichnet (siehe Abbildung 4).*

Das IQTIG sammelt die öffentlichen Schlüssel der Datenservices der Beteiligten Institutionen und stellt diese auf der Website des IQTIG unter folgender Adresse zur Verfügung: <https://iqtig.org/datenerfassung/servicedateien/>

Bezüglich der Maßgabe die privaten Schlüssel niemals an Dritte weiterzugeben sei ergänzend angemerkt, dass die XML-Verschlüsselung zur Einhaltung des Datenschutzes erforderlich ist und die Schlüssel ein Hilfsmittel für deren Sicherstellung darstellen. Daher ist insbesondere für den privaten Schlüssel besondere Sorgsamkeit geboten. Dessen Ablage muss gesichert sein und er darf lediglich autorisierten Personen zugänglich sein. Idealer Weise ist der Zugang ohne Passphrase nicht möglich. Für die sichere Ablage kann beispielsweise auf eine Smartcard zurückgegriffen werden. Sollte der private Schlüssel trotz sorgsamem Umgang kompromittiert sein, ist dies unverzüglich nach Kenntnisnahme dem IQTIG mitzuteilen, um einen geregelten, kurzfristigen Schlüsseltausch (insbesondere des öffentlichen Schlüssels) zu initiieren.

## 3.1 Schlüsselformate

Werden RSA Schlüsselpaare über die API des XPack erzeugt, so werden der öffentliche Schlüssel und der private Schlüssel jeweils Base64-kodiert und als Dateien (\*.pri, \*.pub) im Dateisystem abgelegt. Es können allerdings auch RSA Schlüssel verwendet werden, die nicht über den XPack erzeugt wurden. Diese liegen sehr häufig im PEM-Format vor. Bis zur Version 4.2.9 (einschließlich) des Packers kommt es zu einem Fehler, da die Software mit diesem Format nicht umgehen kann. Ab der Version 4.2.10 können Schlüssel ist dies möglich. Das PEM-Format kennzeichnet sich durch einen Header und Footer, die jeweils mit 5 Spiegelstrichen beginnen und enden.

### Beispiel: Public Key (PEM Format)

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w00...DfTL721LfXCi3/Tw/b7MIgjn8rQxrQ8M3/tBAPHxr/W7
-----END PUBLIC KEY-----
```

## 4 Datei-Formate und Prozesse

Grundsätzlich handelt es sich um XML-Dateien, welche vom Leistungserbringer erzeugt werden. Bevor diese jedoch zur Datenannahmestelle transportiert werden können, müssen sie aus Datenschutzgründen bearbeitet werden:

- Zum einen wird in den Dateien eine Verschlüsselung vorgenommen. Verschiedene Bereiche/Elemente werden – wie zuvor beschrieben – mit den öffentlichen Schlüsseln der Empfänger verschlüsselt. Obwohl sich alle Informationen für alle Beteiligten innerhalb einer XML-Datei befinden, ist sichergestellt, dass Unbefugte diese Bereiche/Elemente nicht einsehen können – wohl aber der Empfänger, der die Dateien mit seinem privaten Schlüssel entschlüsselt. Dieser Schritt ist nur notwendig, wenn in der Datei dokumentierte QS-Fälle mit patientenidentifizierenden Daten (PID) enthalten sind.



### Achtung

- Sind keine QS-Daten in der Datei enthalten (wie z.B. bei Protokollen), kann dieser Schritt übersprungen werden.
- Vor und nach der Verschlüsselung/Entschlüsselung handelt es sich immer um XML-Dateien.

- 
- Zusätzlich wird die Datei noch für den Transport per Email „verpackt“. Hierbei wird die Gesamtdatei mit einem symmetrischen Verfahren verschlüsselt. Dazu werden die Registrierungsnummer und das Transport-Passwort benötigt, welches der Absender im Zuge der Registrierung beim Empfänger erhalten hat. Der TPACKER bietet auch die Möglichkeit, die Transportverschlüsselung im K-Mode (Erläuterung siehe Anfang Kapitel 6) mit einem öffentlichen Schlüssel vorzunehmen. Der GPacker unterstützt diese Funktion auch.



### Achtung

- Dieser Schritt ist immer auszuführen, wenn Dateien per Internet oder auf anderen unsicheren Kanälen transportiert werden.
  - Die verpackten Dateien werden im binären AES-Format gespeichert, welches eine sichere Verpackung auf unsicheren Wegen darstellt.
-

## 5 Allgemeines zu Java-Programmen

Kommt es bei Ausführung von Java-Programmen zu einem `java.lang.OutOfMemoryError`, so reicht der per Standard allokierte Arbeitsspeicher zur Verarbeitung des Java-Programms nicht aus. Die etablierten Parameter<sup>[1][2]</sup> zur Steuerung hierzu sind:

```
-Xms<size> set initial Java heap size  
-Xmx<size> set maximum Java heap size
```

Bspw.:

```
java -jar *Packer*.jar <Parameter> -Xmx8192m
```

[1] <https://docs.oracle.com/en/java/javase/14/docs/specs/man/java.html#extra-options-for-java>

[2] <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html#BABHDABI>

## 6 TPACKER

Der TPACKER ist die Konsolenanwendung für die Anwendung der Transportverschlüsselung. Der Programmaufruf erfolgt mittels folgender Syntax:

```
java -jar TPACKER-Cli-<Version>-SNAPSHOT-jar-with-dependencies.jar  
<Funktionsparameter> <Optionsparameter>
```

Grundsätzliches: Der TPACKER stellt zwei verschiedene Modi zur Durchführung der Transportverschlüsselung bereit, den P-Mode und den K-Mode. Der P-Mode steht für das Verschlüsseln mit Passwort. Dabei wird die Datei symmetrisch mit einem zuvor festgelegten Passwort verschlüsselt. Dieses Passwort wird in der Regel bei der Registrierung der im Datenfluss folgenden datenentgegennehmenden Institution vergeben. Der K-Mode steht für das Verschlüsseln mit einem öffentlichen Schlüssel (K-Mode=Key-Mode). Hierbei wird die Datei mit einem Sitzungsschlüssel symmetrisch verschlüsselt und anschließend wird der Sitzungsschlüssel mit dem öffentlichen Schlüssel verschlüsselt. Insgesamt handelt es sich beim K-Mode um eine sogenannte hybride Verschlüsselung, wie sie auch beim XPACKER zur Verschlüsselung der einzelnen XML-Elemente zum Einsatz kommt.

### Funktionsparameter:

A Ausgabe der Hilfe

Aufruf	<b>-h</b>
Erläuterung	Ausgabe der Hilfe zum TPACKER

B Ausgabe der Version

Aufruf	<b>-v</b>
Erläuterung	Ausgabe der Programmversion

C Verschlüsselung (P-Mode)

Aufrufalternativen	<b>encrypt, e, encrypt</b>
Erläuterung	Führt P-Mode-Verschlüsselung (symmetrische Verschlüsselung) der Eingabedatei durch.
Optionsparameter	<ul style="list-style-type: none"><li>• <b>-f, -p</b> (verpflichtend)</li><li>• <b>-o, -t, -y</b> (optional)</li></ul>
Beispiel	<b>java -jar TPACKER-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -e -f file -o outpath -p password</b>

#### D Entschlüsseln (P-Mode)

Aufrufalternativen	<code>decrypt, -d, --decrypt</code>
Erläuterung	Führt Entschlüsselung P-Mode-verschlüsselter Eingabedateien durch.
Optionsparameter	<ul style="list-style-type: none"> <li>• <code>-f, -p</code> (verpflichtend)</li> <li>• <code>-o, -t, -c</code> (optional)</li> </ul>
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -d -f file.aes -o outpath -p password</code>

#### E Verschlüsseln (K-Mode)

Aufrufalternativen	<code>kmodeencrypt, -a, --kmodeencrypt</code>
Erläuterung	Verschlüsselt Eingabedateien im K-Mode (hybrid)
Optionsparameter	<ul style="list-style-type: none"> <li>• <code>-f, -k</code> (verpflichtend)</li> <li>• <code>-o, -t, -y</code> (optional)</li> </ul>
Beispiel	<ul style="list-style-type: none"> <li>• <code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -a -f file -o outpath -k key.pub</code></li> </ul>

#### F Entschlüsseln (K-Mode)

Aufrufalternativen	<code>kmodedecrypt, -b, --kmodedecrypt</code>
Erläuterung	Entschlüsselung von Dateien, die im K-Mode (hybrid) verschlüsselt worden sind.
Optionsparameter	<ul style="list-style-type: none"> <li>• <code>-f</code> (verpflichtend)</li> <li>• <code>-o, -t</code> (optional)</li> <li>• Alternativen <ul style="list-style-type: none"> <li>◦ Bei Verwendung eines privaten Schlüssels aus einer Schlüssel-datei: Option <code>-k</code></li> <li>◦ Bei Verwendung eines privaten Schlüsseleintrages aus einem PKCS12-Keystore: Optionen: <code>-l, -q, -r, -s</code></li> </ul> </li> </ul>
Beispiele	<ul style="list-style-type: none"> <li>• <code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -b -f file.aes -o outpath -k key.pri</code></li> <li>• <code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -b -f file.aes -o outpath -s store.p12 -l keyalias -q storepassword -r en-trypassword</code></li> </ul>

## G Komprimieren

Aufrufalternativen	<code>zip, -z, --zip</code>
Erläuterung	Komprimierung von Dateien im Zip-Format
Optionsparameter	<ul style="list-style-type: none"><li>• <code>-f</code> (verpflichtend)</li><li>• <code>-o, -t</code> (optional)</li></ul>
Beispiel	<code>java -jar TPacker-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -z -f file -o outfile</code>

## H Dekomprimieren

Aufrufalternativen	<code>unzip, -u, --unzip</code>
Erläuterung	Dekomprimierung von Dateien im Zip-Format
Optionsparameter	<ul style="list-style-type: none"><li>• <code>-f</code> (verpflichtend)</li><li>• <code>-o</code> (optional)</li></ul>
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -u -f file.zip -o outpath</code>

## I Komprimieren und P-Mode-Verschlüsseln (symmetrisch)

Aufrufalternativen	<code>zipencrypt, -ze, --zipencrypt</code>
Erläuterung	Komprimieren und P-Mode-Verschlüsseln (symmetrisch) von Dateien
Optionsparameter	<ul style="list-style-type: none"><li>• <code>-f, -p</code> (verpflichtend)</li><li>• <code>-o, -t, -y</code> (optional)</li></ul>
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -ze -f file -o outfile -p password</code>

## J Entschlüsseln (P-Mode) und Dekomprimieren

Aufrufalternativen	<code>decryptunzip, -du, --decryptunzip</code>
Erläuterung	Entschlüsseln und Dekomprimieren von Dateien, die P-Mode-verschlüsselt (symmetrisch) worden sind.
Optionsparameter	<ul style="list-style-type: none"><li>• <code>-f, -p</code> (verpflichtend)</li><li>• <code>-o, -c</code> (optional)</li></ul>
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -du -f file.zip.aes -o out-path -p password</code>

## K Komprimieren und K-Mode-Verschlüsseln (hybrid)

Aufrufalternativen	<code>zipmodeencrypt, -za, --zipmodeencrypt</code>
Erläuterung	Komprimieren und Verschlüsseln im K-Mode (hybrid) von Dateien.
Optionsparameter	<ul style="list-style-type: none"> <li>• <code>-f, -k</code> (verpflichtend)</li> <li>• <code>-o, -t, -y</code> (optional)</li> </ul>
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -za -f file -o outfile -k key.pub</code>

#### L Entschlüsseln (K-Mode) und Dekomprimieren

Aufrufalternativen	<code>kmodedecryptunzip, -bu, --kmodedecryptunzip</code>
Erläuterung	Entschlüsseln und Dekomprimieren von Dateien, die komprimiert und im K-Mode verschlüsselt worden sind.
Optionsparameter	<ul style="list-style-type: none"> <li>• <code>-f</code> (verpflichtend)</li> <li>• <code>-o</code> (optional)</li> <li>• Alternativen: <ul style="list-style-type: none"> <li>◦ Bei Verwendung eines privaten Schlüssels aus einer Schlüssel-datei: Option <code>-k</code></li> <li>◦ Bei Verwendung eines privaten Schlüsseleintrages aus einem PKCS12-Keystore: Optionen: <code>-l, -q, -r, -s</code></li> </ul> </li> </ul>
Beispiele	<ul style="list-style-type: none"> <li>• <code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -bu -f file.zip.aes -o out-path -k key.pri</code></li> <li>• <code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -bu -f file.zip.aes -o out-path -s store.p12 -l keyalias -q storepassword -r entrypassword</code></li> </ul>

#### M Prüfen des Verschlüsselungstyps

Aufrufalternativen	<code>checkencryptiontype, -c, --checkencryption-type</code>
Erläuterung	Mit dieser Funktion kann geprüft werden, ob es sich um eine Datei handelt, die im K-Mode verschlüsselt worden ist. In diesem Fall wird „KMODE“ ausgegeben. Falls nicht, wird „NONKMODE“ ausgegeben.
Optionsparameter	<code>-f</code> (verpflichtend)
Beispiel	<code>java -jar TPacker-Cli-&lt;Version&gt;-SNAPSHOT-jar-with-dependencies.jar -c -f file.aes</code>

**Bemerkung:** Die Syntax der bestehenden Befehle (P-Mode) hat sich mit der Einführung des neuen Kommandozeilen-Interfaces geringfügig geändert, so dass sich einzelne Befehle nicht mehr kombinieren lassen. Um beispielsweise eine Datei im P-Mode zu komprimieren und zu

verschlüsseln muss die Funktion

`-ze` angegeben werden. Die Kombination der Befehle `-z` und `-e` in der Form `java -jar TPacker-Cli-<Version>-SNAPSHOT-jar-with-de-pendencies.jar -z -e ...` ist nicht möglich.

#### Optionsparameter:

- i. Option `-f, --input_files=<files>`  
Spezifiziert die Eingabedatei(en). Falls mehrere Dateien verarbeitet werden sollen, müssen diese durch Kommata ohne Leerschritt getrennt werden.
- ii. Option `-o, --output=<output>`  
Pfad für den Output der entsprechenden Funktion.
- iii. Option `-p, --password=<password>`  
Passwort für die (symmetrische) P-Mode-Verschlüsselung/-Entschlüsselung.
- iv. Option `-t, --timestamp`  
Wenn diese Option gesetzt ist, wird kein Zeitstempel (TimeStamp) im Ausgabedateinamen erzeugt.
- v. Option `-y, --unsafe`  
Überspringt die eingebaute Prüfung, ob VERSICHERTENID im Dokument vorhanden ist.
- vi. Option `-c, --characterset_pw_decrypt=<characterSet>`  
Optionale Angabe zur Zeichencodierung des Passworts zur Entschlüsselung, bspw. 'UTF-8' oder 'ISO-8859-1'. Nur gedacht zur Lösung von Entschlüsselungsproblemen mit Altdateien (verschlüsselt mit Version < 4.2.11) in Kombination mit Sonderzeichen in Passwörtern.
- vii. Option `-k, --keyfile=<keyFile>`  
Pfad zu Datei mit öffentlichem Schlüssel bei K-Mode-Verschlüsselung bzw. zu Datei mit privatem Schlüssel bei K-Mode-Entschlüsselung.
- viii. Option `-s, --pkcs12file=<pkcs12File>`  
Bei der Entschlüsselung kann statt einem keyfile auch mit dieser und den drei folgenden Optionen ein Schüsseleintrag in einem PKCS12-Keystore angegeben werden. Diese Option dient dabei der Angabe des Pfades zu der Keystore-Datei.
- ix. Option `-q, --storepasswd=<keystorePassword>`  
Bei der Angabe eines Keystore-Eintrages für den privaten Schlüssel wird mit dieser Option das Passwort für den Keystore angegeben.
- x. Option `-l, --keyalias=<keyentryAlias>`  
Bei der Angabe eines Keystore-Eintrages für den privaten Schlüssel wird mit dieser Option der Alias zu dem Schlüsseleintrag angegeben.
- xi. Option `-r, --keypasswd=<keyentryPassword>`  
Bei der Angabe eines Keystore-Eintrages für den privaten Schlüssel wird mit dieser Option das Passwort des entsprechenden Schlüsseleintrages angegeben.

# 7 XPacker

Der XPacker ist die Konsolenanwendung für die Anwendung der XML-Verschlüsselung. Der Programmaufruf erfolgt mittels folgender Syntax:

```
java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar  
[Parameter]
```

## Basisfunktionen

### Parameter:

- h      gibt die Hilfe aus.
- e      Die Eingangsdateien werden verschlüsselt.
- d      Die Eingangsdateien werden entschlüsselt.
- g      Es wird ein öffentlicher und ein privater Schlüssel erzeugt. Standardnamen key.pub, key.pri
- f      Eingangsdatei, falls keine angegeben wird die Standardeingabe verwendet.
- o      Ausgabedatei, falls keine angegeben wird alles an die Standardausgabe geleitet.
- k      Beim Verschlüsseln wird der öffentliche Schlüssel benötigt und beim Entschlüsseln der private Schlüssel.
- t      Liste der XML-Tags, die mit dem Schlüssel ver-/entschlüsselt werden.
- 2      Gibt den Namen eines XML-Tags an, unter dem im Header der XML-Datei nach der Verschlüsselung die KeyInfo-Elemente abgelegt werden. Ein solches KeyInfo-Element enthält die kryptografischen Schlüssel, die bei der Verschlüsselung zum Einsatz kamen. Nämlich einerseits den AES-Schlüssel und andererseits Modulus und Exponent des öffentlichen RSA-Schlüssels des Empfängers. Wird das "-2"-Flag nicht verwendet, wird das entsprechende Tag standardmäßig "encryption" genannt.

### Beispiele:

- `java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -e -f inputfile -o outputfile -k keyfile -t tag1,tag2... -2 tagForEncryptedKey`
- `java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -e -k keyfile -t tag1,tag2... -2 tagForEncryptedKey < inputStream > outputStream`
- `java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -d -f inputfile -o outputfile -k privateKeyFile -t tag1,tag2...`
- `java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -d -k keyfile -t tag1,tag2... < inputStream > outputStream`
- `java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -g -k keyname`

# Ergänzende Funktionen

## 1. Ausgabe des Modulus eines öffentlichen Schlüssels

Mit der Angabe des Funktionsflag `-m` (Kurzform) bzw. `--getmodulus` (Langform) kann der Modulus eines mit der Option `-k` zu übergebenden öffentlichen Schlüssels ausgegeben werden.

### Beispiel-Aufruf:

```
java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -m -k keyfile.pub
```

Dieser Aufruf gibt den Modulus des Schlüssel der Datei `keyfile.pub` aus.

### Beispiel-Ausgabe:

```
q0tUGw0qehjM8HcAB2A3RsMibCOBh6lhtIcNmIwAGDbSIPcHe6gprgxcCTHs9VYTdQHYxOABUwvaWtNp
FS5Z7JFXb1l7IquYc8H84pCcV5LqddycA+pibexi7g/VqKzvRLeWtkWeArGHman9BcSAPHav4e4rhHC
PIKEL2Y8SJNK0Ru01f5oq12fzmg5fQQFZH1Xiztp59DcdYXg74yZa/RCsAprS7dhjzuYW+rqSmRI6TW8
P5qskQ/T0s7Zyh4i/EEVZKVSyaClJr6AzsZdri1e+9VNyf++k5d5HtXl1DZ3v2yg+AymGJ4w+DUjN9Oo
6XIEDFraFWWhWyz7iZiilVQ==
```

**Bemerkung:** Der Modulus ist die Base-64-Darstellung eines Big-Integer-ByteArrays, allerdings ohne ein Vorzeichen-Byte. Andere Tools/Bibliotheken stellen die Zahl mit einem 0-Byte als Vorzeichen-Byte dar, so dass sich das Ergebnis von dem anderer Tools/Bibliotheken unterscheiden kann.

## 2. Prüfung der Verschlüsselung eines XML-Tags hinsichtlich des Einsatzes eines bestimmten öffentlichen Schlüssels

Hat man eine XML-Datei mit verschlüsselten Tags vorliegen, kann man mit dieser Funktion prüfen, ob bei bestimmten XML-Tags ein bestimmter Schlüssel zum Einsatz gekommen ist. Dazu ruft man den XPacker mit dem Funktionsflag `-y` (Kurzform) bzw. `--checkkey` (Langform) auf, übergibt mit der Option `-k` die Datei mit dem öffentlichen Schlüssel und spezifiziert das Tag, dessen Verschlüsselung geprüft werden soll, mit der Option `-t`. Außerdem muss die XML-Datei mit der Option `-f` übergeben werden.

### Beispiel-Aufruf:

```
java -jar XPacker-<Version>-SNAPSHOT-jar-with-dependencies.jar -y -f  
beispiel.xml -k keyfile.pub -t patient
```

Dieser Aufruf prüft, ob in der Datei `beispiel.xml` das Tag `patient` mit dem Schlüssel in der Datei `keyfile.pub` verschlüsselt worden ist.

### Beispiel-Ausgabe:

```
TRUE (Schluessel wurde für die Verschluesselung von Tag 'patient' verwendet.)
```

### Bemerkung:

Möchte man prüfen, wo z.B. in einer ganzen Menge von XML-Dateien ein bestimmter öffentlicher Schlüssel zum Einsatz kam, kann man sich auch den Modulus mit der vorherigen Funktion ausgeben lassen und danach (oder nach einem Teilstring) in den XML-Dateien suchen. Denn bei jeder XML-Tag-Verschlüsselung wird der Modulus in der XML-Datei mit abgelegt.

## 8 GParser

Für alle Anwender, die händisch die Verschlüsselung durchführen müssen, stellt der GParser mit seiner grafischen Oberfläche eine interaktive Alternative zur Verwendung der Programme XParser und TParser dar. Die beiden Programme werden nicht benötigt, wenn der GParser eingesetzt wird.

Der GParser umspannt drei Kategorien von Funktionen, die sich in den drei auf der Anwendungsoberfläche existierenden Tabs *P-Mode*, *K-Mode* und *Schlüssel* widerspiegeln:

- *Schlüssel*: Auf diesem Tab können die privaten und öffentlichen Schlüssel verwaltet werden. Hier können aus Schlüsseldateien private und öffentliche Schlüssel eingelesen und gespeichert werden. Jeder Schlüssel, der dann für die XML-Inhaltsverschlüsselung im P-Mode und die K-Mode-Verschlüsselung verwendet werden soll, muss in der Schlüsselverwaltung vorliegen, also mindestens einmal eingelesen worden sein.
- *P-Mode*: Die Funktionen auf diesem Tab entsprechen weitestgehend den Funktionen, die der GParser vor Version 4.4.0 besaß, d.h. dem hybriden inhalts- und symmetrischen Transportverschlüsseln von XML-Dateien. Außerdem können Zertifikatsdateien (.cer)-Dateien für den Transport symmetrisch mit Passwort verschlüsselt werden.
- *K-Mode*: Dieser Tab stellt die Funktion bereit, XML-Dateien oder Zertifikatsdateien (.cer)-Dateien hybrid im K-Mode d.h. mit einem öffentlichen bzw. privaten Schlüssel zu verschlüsseln bzw. entschlüsseln.

Im folgenden werden die Funktionen der drei Kategorien im Detail erklärt.

### 8.1 Schlüsselverwaltung

#### 8.1.1 Grundprinzip

Für Inhaltsver- bzw. Inhaltsentschlüsselung des P-Modus und die K-Mode-Ver- bzw. K-Mode-Entschlüsselung werden öffentliche bzw. private Schlüssel benötigt, die unter dem Tab Schlüssel eingelesen werden bzw. verwaltet werden können. Beim Einlesen werden die Schlüssel mit Namen versehen, so dass sie dann in den Drop-down-Menüs im P-Mode und im K-Mode unter diesem Namen ausgewählt werden können. Die öffentlichen Schlüssel werden dauerhaft gespeichert, so dass sie bei der nächsten Sitzung nicht erneut eingelesen werden müssen. Auch die privaten Schlüssel können dauerhaft gespeichert werden, wenn sie zuvor zum Schutz vor unberechtigt Zugriff mit einem Passwort versehen wurden.

## 8.1.2 Bedienelemente

Abbildung 2: Bedienelemente der Schlüsselverwaltung

The screenshot shows a web-based key management interface. At the top, there are tabs for 'P-Mode', 'K-Mode', and 'Schlüssel'. Below this, a radio button labeled 'S01' allows switching between 'öffentlich' (public) and 'privat' (private) key management. A dropdown menu labeled 'S02' shows 'Hinzugefügte Schlüssel' with a selection of '< Keine Auswahl >'. A status message 'S04' indicates 'Status: Insgesamt 0 öffentliche Schlüssel vorhanden.' A list of keys is shown with callouts S06, S04, S08, S09, S10, and S11. On the right, a 'Löschen' button is labeled S03. Below the key list, a 'Schlüsseldetails:' section contains fields for 'Schlüsseldatei-Typ', 'Schlüsseldatei', 'Passwort Keystore', 'Schlüssel-Alias', and 'Passwort Schlüssel'. A 'Neu' button is labeled S05. A 'Hinzugefügte Schlüssel' dropdown is labeled S07. A 'Schlüsselname' field is labeled S12. A 'Hinzufügen' button is labeled S13. At the bottom, there are buttons for 'Private Schlüssel sperren' (S14), 'Passwort' (S15), 'Passwort Wdh.' (S16), 'Passwort zurücksetzen' (S18), and 'Private Schlüssel verwerfen' (S19).

1. **S01** Radio-Button "Verwaltet Schlüsselart": An diesem Radiobutton kann gewählt werden, ob aktuell öffentliche oder private Schlüssel verwaltet werden sollen. Wechselt man beispielsweise von privat auf öffentlich, ist dann unter S02 nicht mehr die private Schlüsselliste, sondern die öffentliche verfügbar.
2. **S02** Drop-down-Menü "Hinzugefügte Schlüssel": An diesem Drop-down-Menü kann, wenn bereits Schlüssel hinzugefügt wurden, ein Schlüssel ausgewählt werden, um die Schlüsseldetails anzuzeigen oder um ihn mit Knopf S03 zu löschen.
3. **S03** Knopf "Löschen": Nachdem über S02 ein Schlüssel ausgewählt wurde, kann er über diesen Knopf gelöscht werden.
4. **S04** Textlabel "Status": Zeigt den aktuellen Status der Schlüsselliste an, z.B. wieviele Schlüssel der unter S01 gewählten Kategorie bereits hinzugefügt wurden. Außerdem weist dieses Label daraufhin, wenn bei Auswahl von "privat" unter S01 die gespeicherten privaten Schlüssel entsperrt oder (falls z.B. das Passwort vergessen wurde) verworfen werden müssen.
5. **S05** Knopf "Neu": Auf diesen Knopf klickt man, wenn man einen neuen Schlüssel der Schlüsselliste hinzufügen möchte.
6. **S06** Drop-down-Menü "Schlüsseldatei-Typ": Hier kann man auswählen, von welchem Typ von Schlüsseldatei man den neuen Schlüssel einlesen möchte. Im Falle, dass man einen öffentlichen importieren möchte (dementsprechend unter S01 "öffentlich" ausgewählt ist), ist hier die Option "Schlüsseldatei" vorausgewählt. Im Falle eines neuen privaten Schlüssels steht hier zusätzlich die Option "Schlüsselcontainer (PKCS 12)" zur Verfügung, so dass bei dieser Option der private Schlüssel aus einem PKCS12-Schlüsselcontainer importiert werden kann.
7. **S07** Knopf "...": Über diesen Knopf kann die Schlüsseldatei bzw. Schlüsselcontainer-Datei ausgewählt werden, aus der der Schlüssel importiert werden soll.
8. **S08** Feld "Schlüsseldatei/Schlüsselcontainer": Hier wird der Pfad der über S07 ausgewählten Schlüssel- bzw. Containerdatei angezeigt.

S09 - S11 sind nur relevant beim Einlesen von privaten Schlüsseln aus einer Keystoredatei (PKCS12):

9. **S09** Feld "Passwort Keystore": Hier ist das Passwort anzugeben, mit dem die Keystoredatei gesichert ist, aus der der private Schlüssel eingelesen werden soll.
10. **S10** Feld "Schlüssel-Alias": Hier ist der Alias des Schlüsseleintrag der Keystoredatei (PKCS12) anzugeben, aus dem der private Schlüssel importiert werden soll.
11. **S11** Feld "Passwort Schlüssel": Hier muss das Passwort angegeben werden, mit dem der Schlüsseleintrag der Keystoredatei (PKCS12) geschützt ist, aus dem der private Schlüssel importiert werden soll.
12. **S12** Feld "Schlüsselname": Hier wird der Schlüsselname angegeben, unter dem der Schlüssel mit den Details S06,S08, ggf. S09-S11 in der Schlüsselliste abgespeichert werden soll.
13. **S13** Knopf "Hinzufügen": Hat man die Schlüsseldetails S06, S08, ggf. S09-S11 angegeben, wird der Schlüssel mit dem Klick auf diesen Knopf eingelesen.

Die folgenden Bedienelemente sind relevant für die Verwaltung der privaten Schlüsselliste:

14. **S14** Schlosssymbol: Zeigt an, ob die Schlüsselliste gesperrt oder entsperrt ist.
15. **S15** Knopf "Private Schlüssel sperren/entsperren": Über diesen Knopf wird die Schlüsselliste gesperrt oder entsperrt.
16. **S16** Feld "Passwort": Beim erstmaligen Sperren der privaten Schlüsselliste bzw. beim Entsperren wird hier ein Passwort angegeben.
17. **S17** "Passwort Wdh.": Beim erstmaligen Sperren oder beim Sperren, nachdem das Passwort zurückgesetzt worden ist, muss hier die Wiederholung des Passwortes eingegeben werden.
18. **S18** Knopf "Passwort zurücksetzen": Ist die private Schlüsselliste entsperrt, kann über diesen Knopf das Passwort zurückgesetzt werden und im Anschluss neu vergeben werden.
19. **S19** Knopf "Private Schlüssel verwerfen": Sind die privaten Schlüssel gesperrt, können mit diesem Knopf die bestehenden Schlüssel verworfen werden, z.B. weil das zum Sperren verwendete Passwort vergessen wurde. Die privaten Schlüssel müssen danach wieder neu eingelesen werden.

## 8.1.3 Verwaltung öffentlicher Schlüssel

Um die öffentlichen Schlüssel zu verwalten, muss zunächst auf dem Tab "Schlüssel" der Radiobutton "öffentlich" (S01) ausgewählt werden.

### Hinzufügen öffentlicher Schlüssel

Öffentliche Schlüssel aus einer Schlüsseldatei werden folgendermaßen zur öffentlichen Schlüsselliste hinzugefügt:

1. Klick auf den Knopf "Neu" (S05).
2. Jetzt sollte in dem Drop-down-Menü "Schlüsseldatei-Typ" (S06) der Eintrag "Schlüsseldatei" vorausgewählt sein.
3. Über Knopf "..." (S07) wird die Schlüsseldatei, die den zu importierenden Schlüssel enthält, ausgewählt.
4. Im Feld "Schlüsseldatei" (S08) steht jetzt der Pfad der ausgewählten Schlüsseldatei.
5. Als nächstes vergibt man in dem Feld "Schlüsselname" (S12) einen Namen, unter dem der Schlüssel künftig in dem Drop-down-Menü "Hinzugefügte Schlüssel" und auch auf den anderen Tabs "P-Mode" bzw. "K-Mode" in den Schlüssel-Drop-down-Menüs erscheinen soll. Dieser Name muss eindeutig sein, d.h. es können nicht zwei öffentliche Schlüssel mit gleichem Namen benannt werden.
6. Nun klickt man auf den Knopf "Hinzufügen" (S13).
7. Nach erfolgreichem Hinzufügen sollte die Statusmeldung (S04) "Öffentlicher Schlüssel <Schlüsselname> hinzugefügt. Insgesamt <n> öffentliche Schlüssel vorhanden." lauten. Wobei n sich gegenüber der Meldung vor dem Hinzufügen des Schlüssels sich um eins erhöht haben sollte. In dem Drop-down-Menü "Hinzugefügte Schlüssel" (S02) erscheint jetzt der hinzugefügte Schlüssel und ist vorausgewählt. Im P-Mode oder K-Mode erscheint jetzt der Schlüssel unter seinem Namen in den Drop-down-Menüs mit den Schlüsseln zur Verschlüsselung (P09 bzw. K04). Dies wird sichtbar, wenn man im K-Mode bzw. P-Mode jeweils die Aktion Verschlüsseln ausgewählt hat, und man sich den Inhalt der Drop-down-Menüs (P09 bzw. K04) anzeigen lässt.

### Anzeigen öffentlicher Schlüssel

Hat man einen oder mehrere Schlüssel der Schlüsselliste hinzugefügt, können diese über das Drop-down-Menü "Hinzugefügte Schlüssel" (S02) ausgewählt werden. Es werden dann die Schlüsseldetails, in diesem Fall der Pfad der Schlüsseldatei, aus dem der Schlüssel importiert wurde, in S08 angezeigt. Nachdem der Schlüssel eingelesen wurde, ist es nicht mehr notwendig, dass die Datei an dieser Stelle fortbesteht. Der Pfad wird unter S08 nur zur Erinnerung angezeigt, um sichtbar zu machen, woher der Schlüssel ursprünglich stammte, und hat darüberhinaus keine Funktion.

## Löschen von öffentlichen Schlüsseln

Um einen öffentlichen Schlüssel zu löschen, geht man wie folgt vor:

1. Man wählt den Schlüssel unter seinem Namen in dem Drop-down-Menü "Hinzugefügte Schlüssel" (S02) aus.
2. Man klickt auf den Knopf "Löschen" (S03).
3. Nach erfolgreichem Löschen sollte die Statusmeldung "Öffentlicher Schlüssel '<Schlüsselname>' gelöscht. Insgesamt <n> öffentliche Schlüssel vorhanden." lauten. Dabei soll n um eins niedriger sein, verglichen mit der Statusmeldung vor dem Löschen des Schlüssels.

### 8.1.4 Hinzufügen und Löschen privater Schlüssel

Um private Schlüssel hinzufügen oder löschen zu können, muss zunächst auf dem Tab "Schlüssel" der Radiobutton "privat" (S01) ausgewählt sein. Weitere Voraussetzung für das Hinzufügen oder Löschen privater Schlüssel ist, dass die private Schlüsselliste entsperrt ist, was man daran erkennt, dass das Schlosssymbol S14 grün und geöffnet ist. Ist die private Schlüsselliste gesperrt, ist erstens das Schlosssymbol S14 rot und geschlossen. Zweitens steht im Statusfeld S04 "Entsperren oder verwerfen Sie die privaten Schlüssel.". Das Entsperren und Verwerfen der privaten Schlüssel wird in den noch folgenden Kapiteln beschrieben.

Beim Hinzufügen von privaten Schlüsseln gibt es zwei Fälle. Das Hinzufügen aus einer Schlüsseldatei und das Hinzufügen aus einem Schlüsselcontainer (PKCS12).

#### Hinzufügen von privatem Schlüssel aus Schlüssel-Datei

Vorgang funktioniert analog zu "Hinzufügen öffentlicher Schlüssel" mit Unterschied, dass in dem Drop-down-Menü "Schlüsseldatei-Typ" (S06) der Eintrag "Schlüsseldatei" nicht vorausgewählt ist, sondern aktiv ausgewählt werden muss.

#### Hinzufügen privater Schlüssel aus einem Schlüsselcontainer (PKCS12)

1. Klick auf den Knopf "Neu" (S05).
2. Auswahl des Eintrages "Schlüsselcontainer (PKCS12)" in dem Drop-down-Menü "Schlüsseldatei-Typ" (S06).
3. Über Knopf "..." (S07) wird die Schlüsselcontainer-Datei (.p12) ausgewählt, aus dem der Schlüssel importiert werden soll.
4. Im Feld "Schlüsselcontainer" (S08) steht jetzt der Pfad der ausgewählten Schlüsselcontainer-Datei.

5. Im Feld "Passwort Keystore" (S09) gibt man nun das Passwort an, mit dem die Schlüsselcontainer-Datei geschützt ist.
6. Im Feld "Schlüssel Alias" (S10) gibt man nun den Alias des Schlüsseleintrages an, aus dem man den privaten Schlüssel einlesen möchte.
7. Im Feld "Passwort Schlüssel" (S11) gibt man nun das Passwort an, mit dem der Schlüsseleintrag geschützt ist.
8. Als nächstes vergibt man in dem Feld "Schlüsselname" (S12) einen Namen, unter dem der Schlüssel künftig in dem Drop-down-Menü "Hinzugefügte Schlüssel" und auch auf den anderen Tabs "P-Mode" bzw. "K-Mode" in den Schlüssel-Drop-down-Menüs (P09 bzw. K04) erscheinen soll. Dieser Name muss eindeutig sein, d.h. es können nicht zwei private Schlüssel mit gleichem Namen benannt werden.
9. Nun klickt man auf den Knopf "Hinzufügen" (S13).
10. Analog zum Vorgang "Hinzufügen öffentlicher Schlüssel" aktualisiert sich nach erfolgreichem Hinzufügen die Statusmeldung S04. In dem Drop-down-Menü "Hinzugefügte Schlüssel" (S02) erscheint jetzt der hinzugefügte Schlüssel und ist vorausgewählt. In den Schlüssel-Drop-down-Menüs (P09 bzw. K04) im P-Mode und K-Mode kommt nun bei jeweiliger Auswahl der Aktion "Entschlüsseln" der eben hinzugefügte Schlüssel vor.

### **Löschen privater Schlüssel**

Funktioniert analog zu Löschen von öffentlichen Schlüsseln.

## 8.1.5 Sonstige Verwaltung der privaten Schlüssel

Die Verwaltung öffentlicher Schlüssel und privater Schlüssel unterscheidet sich darin, dass die öffentlichen Schlüssel automatisch gespeichert werden, während die privaten Schlüssel nur gespeichert werden, wenn sie vorher mit einem Passwort geschützt bzw. gesperrt worden sind. Wir wenden uns nun der Verwaltung der privaten Schlüssel zu. Grundsätzlich gibt es drei Zustände, in dem sich die private Schlüsselliste befinden kann:

- *ZE Schlüsselliste entsperrt, Passwort nicht vergeben.* Diesen Zustand erkennt man daran, dass Schlosssymbol S14 grün und geöffnet ist und die Felder S16 "Passwort" und S17 "Passwort Wdh." beide aktiv (weiß) sind.
- *ZG Schlüsselliste gesperrt.* Diesen Zustand erkennt man daran, dass das Schlosssymbol S14 rot und geschlossen ist.
- *ZEP Schlüsselliste entsperrt, Passwort vergeben.* Diesen Zustand erkennt man daran, dass Schlosssymbol S14 grün und geöffnet ist und die Felder S16 "Passwort" und S17 "Passwort Wdh." beide inaktiv (grau) sind.

Im P-Mode und K-Mode können die privaten Schlüssel erst dann verwendet werden, wenn sich die private Schlüsselliste im Zustand ZE oder ZEP befindet. Vorher sind die Drop-down-Menüs P09 (P-Mode) bzw. K04 (K-Mode) mit den für die Entschlüsselung bereitstehenden privaten Schlüsseln entsprechend leer.

Die folgenden Verfahren zum Verwalten der privaten Schlüssel können auf allen Tabs vorgenommen werden, da die Bedienelemente S15 – S19 auf allen Tabs vorhanden sind.

**Wichtig zu beachten:** Verlässt man das Programm im Zustand ZE, werden die privaten Schlüssel nicht gespeichert und müssen bei der nächsten Sitzung erneut importiert werden.

### Sperren der privaten Schlüsselliste bei noch nicht vergebenem Passwort

Dass das Passwort noch nicht vergeben wurde, bedeutet, dass man sich in Zustand ZE befindet. Man sperrt dann die Schlüsselliste wie folgt:

- Man gibt in die Felder S16 "Passwort" und S17 "Passwort Wdh." jeweils das gleiche Passwort ein und klickt auf den Knopf S15 "Private Schlüssel sperren".
- Als Ergebnis sieht man, dass die Schlüsselliste nun gesperrt ist: Das Schlosssymbol S14 ist rot und geschlossen. (Zustand ZG)

### Entsperren der privaten Schlüsselliste

Befindet sich die Schlüsselliste in Zustand ZG und möchte man die Schlüsselliste entsperren, geht man dabei wie folgt vor:

- Man gibt in Feld S16 "Passwort" das Passwort ein, mit dem die Schlüsselliste zuvor gesperrt worden ist.
- Man klickt auf den Knopf S15 "Private Schlüssel entsperren".

- Als Ergebnis sieht man, dass die Schlüsselliste nun entsperrt ist und sich im Zustand ZEP befindet.

### **Sperren der privaten Schlüsselliste bei bereits vergebenem Passwort**

Befindet man sich im Zustand ZEP, nachdem die private Schlüsselliste zuvor entsperrt worden ist, kann man sie wieder mit dem gleichen Passwort, mit dem zuvor gesperrt worden ist, sperren, indem man auf den Knopf S15 "Private Schlüssel sperren" klickt. Als Ergebnis befindet man sich wieder in Zustand ZG.

### **Zurücksetzen des Passwortes**

Hat man die private Schlüsselliste zuvor entsperrt, kann man das Passwort zurücksetzen, indem man auf den Knopf S18 "Passwort zurücksetzen" klickt. Man befindet sich nun wieder in Zustand ZE. Möchte man nun sicherstellen, dass die privaten Schlüssel bis zur nächsten Sitzung gespeichert werden, muss vor dem Schließen, wie unter "Sperren der privaten Schlüsselliste bei noch nicht vergebenem Passwort" beschrieben, die private Schlüsselliste wieder mit einem neuen Passwort gesperrt werden.

### **Verwerfen der privaten Schlüsselliste**

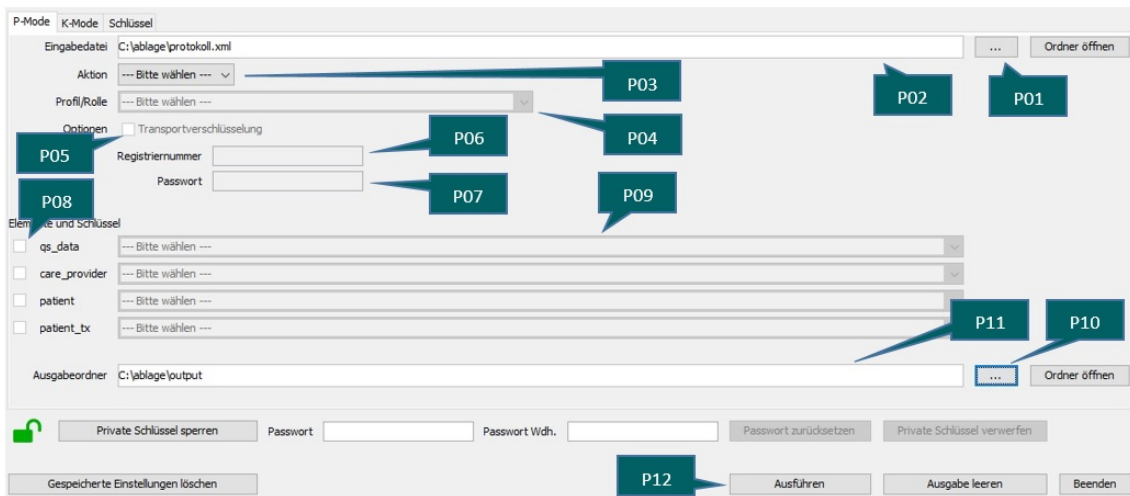
Hat man das Passwort zum Entsperren vergessen, bleibt einem nichts anderes übrig, als die gespeicherte private Schlüsselliste zu verwerfen. Dies geschieht, indem man im Zustand ZG auf den Knopf S19 "Private Schlüssel verwerfen" klickt. Danach ist man wieder in der Lage neue Schlüssel der privaten Schlüsselliste hinzuzufügen.

## **8.2 P-Mode**

Im P-Mode können XML-Dateien, die im Rahmen der DeQS-RL oder oKFE-RL erstellt worden sind, hybrid inhaltsverschlüsselt und entschlüsselt werden, d.h. es werden innerhalb des XMLs bestimmte XML-Elemente, die von der ausgewählten Rolle/Profil abhängen, verschlüsselt bzw. entschlüsselt. Zusätzlich kann die XML-Datei mit einem Passwort transportverschlüsselt werden, so dass im Gegensatz zu der reinen Inhaltsverschlüsselung, bei der wieder eine XML-Datei resultiert, eine komprimierte und symmetrisch verschlüsselte (.zip.aes)-Datei ausgegeben wird. Diese Funktionen entsprechen weitestgehend den Funktionen des GPackers vor Version 4.4.0 Mit dem Unterschied, dass jetzt nicht mehr Schlüsseldateien als Quelle für die Schlüssel angegeben werden, sondern über die Dropdown-Menüs P09 die Schlüssel ausgewählt werden können, die zuvor über die Schlüsselverwaltung importiert worden sind. Außerdem besitzt der P-Mode nun auch die Funktion, Zertifikats-Dateien (.cer) nur rein symmetrisch mit Passwort zu verschlüsseln.

## 8.2.1 Bedienelemente

Abbildung 3: Bedienelemente P-Mode



1. **P01** Knopf "...": Über diesen Knopf wird die zu ver- bzw. entschlüsselnde (.xml)(.cer) bzw. (.zip.aes)-Datei ausgewählt
2. **P02** Feld "Eingabedatei": In diesem Feld erscheint der Pfad, der über P01 ausgewählt worden ist.
3. **P03** Drop-down-Menü "Aktion": Über dieses Drop-down-Menü kann eine der Aktionen "Verschlüsseln" oder "Entschlüsseln" ausgewählt werden.
4. **P04** Drop-down-Menü "Profil/Rolle": Hier kann das für die Ver- bzw. Entschlüsselung zu verwendende Profil ausgewählt werden. Hiervon hängt ab, welche XML-Elemente (siehe P08) zu Ver- bzw. Entschlüsselung vorausgewählt werden.
5. **P05** Checkbox "Transportverschlüsselung": Über diese Checkbox wird gesteuert, ob die zu verschlüsselnde Datei transportverschlüsselt wird.
6. **P06** Feld "Registriernummer": In diesem Feld gibt man die Registriernummer an, die Teil des Dateinamens werden soll.
7. **P07** Feld "Passwort": In diesem Feld wird das Passwort angegeben, das man für die symmetrische Transportver- bzw. Transportentschlüsselung verwenden möchte.
8. **P08** Checkboxes für Aktivierung von Inhaltsver- bzw. Inhaltsentschlüsselung für entsprechende XML-Elemente
9. **P09** Drop-down-Menüs zur Auswahl der für die Inhaltsver- bzw. Inhaltsentschlüsselung zu verwendenden Schlüssel. Hier werden die Schlüssel angezeigt, die zuvor über den Tab "Schlüssel" eingelesen und entsprechend benannt worden sind. Je nachdem, ob die unter P03 ausgewählte Funktion "Verschlüsseln" oder "Entschlüsseln" lautet, wird hier die öffentliche oder private Schlüsselliste angezeigt. Sollen hier bei einer letzten Sitzung gespeicherte private Schlüssel verfügbar sein, müssen diese zuerst entsperrt werden.
10. **P10** Knopf "...": Über diesen Knopf wird der Ausgabeordner ausgewählt, in dem die resultierende Datei abgelegt wird.
11. **P11** Feld "Ausgabeordner": Feld zeigt Pfad des über P10 ausgewählten Ordners an.

12. **P12** Knopf "Ausführen": Stößt die Ver- bzw. Entschlüsselung an.

## 8.2.2 Verschlüsseln und Entschlüsseln im P-Mode

### Szenario 1 Versand von QS-Daten

Abbildung 4: Szenario 1 - Versand von QS-Daten

1. Über Knopf P01 wird die zu verschlüsselnde XML-Export-Datei ausgewählt, wie sie aus dem QS-Dokumentationsprogramm exportiert wird.
2. In Feld P02 erscheint der Pfad der ausgewählten Datei.
3. In Drop-down-Menü "Aktion" (P03) wählt man "Verschlüsseln" aus.
4. In Drop-down-Menü "Profil/Rolle" (P04) wählt man das Profil "LE-Leistungserbringer" aus.
5. Man selektiert die Checkbox "Transportverschlüsselung" (P05)
6. Im Feld "Registriernummer" (P06) wird nun die Registriernummer eingeben, die Namensbestandteil der resultierenden Datei werden soll.
7. Im Feld "Passwort" (P07) fügt man das für die Registrierung bilateral ausgetauschte, geheime Passwort zur Transportverschlüsselung ein.
8. Durch die Selektion des speziellen Profils ist die Auswahl der Checkboxes P08 vorbestimmt. Zu jedem selektierten Tag muss nun in dem zugehörigen Drop-down-Menü P09 der passende Schlüssel ausgewählt werden. Konkrete Vorgaben sind der jeweiligen zweckgebundenen Spezifikation zu entnehmen. Beispielsweise muss das Element *qs\_data* in Abhängigkeit zum Verfahren mit dem öffentlichen Schlüssel entweder der zuständigen Datenannahmestelle auf Landesebene oder der Auswertestelle auf Bundesebene verschlüsselt werden.



Die Selektion der Elemente wird durch Ihre Auswahl im Drop-down-Menü "Profil/Rolle" (P04) bestimmt.

9. Über Knopf "..."(P10) wählt man den Ausgabeordner aus, in dem die Ausgabedatei abgelegt werden soll.
10. Schließlich klickt man auf den Knopf "Ausführen" (P12).

**Szenariovariante** Ohne Selektion der Checkbox Transportverschlüsselung werden nur die Elemente der gewählten XML-Datei verschlüsselt, ohne dass dieses Ergebnis wiederum in einer verschlüsselten (.zip.aes)-Datei verpackt wird.

## Szenario 2 Empfang des Rückprotokolls

Abbildung 5: Szenario 2 - Empfang des Rückprotokolls

1. Über Knopf P01 wählt man die Eingabedatei aus, wie sie von der Datenannahmestelle als Datenflussprotokoll verschickt wird. (.zip.aes-Datei)
2. Nun wird automatisch in dem Drop-down-Menü "Aktion" (P03) "Entschlüsseln" gesetzt.
3. In dem Drop-down-Menü "Profil/Rolle" (P04) nimmt man keine Auswahl vor. Denn: Es ist lediglich die Aufhebung der Transportverschlüsselung, nicht jedoch einer XML-(Inhalts-)Entschlüsselung erforderlich, da in Datenflussprotokollen weder das Element *patient* noch das Element *qs\_data* enthalten ist.
4. Im Feld "Passwort" (P07) wird das bei der Registrierung bilateral ausgetauschte, geheime Passwort eingegeben.
5. Über Knopf P10 wählt man den Ausgabeordner aus, in den die entschlüsselte Datei ausgegeben werden soll.
6. Feld P11 zeigt den Pfad des ausgewählten Ausgabeordners an.
7. Man klickt auf den Knopf "Ausführen" (P12).

## Szenariovarianten

1. Wählt man im Drop-down-Menü "Profil/Rolle" (P04) das Profil "Individuell" aus, so wird die Transportdatei entschlüsselt, entpackt und es besteht die Möglichkeit die Verschlüsselung einzeln auswählbarer XML-Elemente unter Verwendung des eigenen

privaten Schlüssels aufzuheben.

2. Durch Auswahl einer XML-Datei mit verschlüsselten Elementen als Eingabedatei (anstelle der gezeigten (.zip.aes)-Datei) können Sie deren Elemente entschlüsseln.

**Wissenswertes** Die angestoßene Entschlüsselung kann, je nach Größe der Datei und Geschwindigkeit Ihres Rechners, mehrere Minuten dauern. Wenn das Entschlüsseln abgeschlossen ist, wird dieses in der Oberfläche dargestellt. Jede erfolgreiche Ausführung mittels Schaltfläche "Ausführen" (P12) erzeugt einen neuen Laufordner.

### Sonderfälle bei Datenlieferungen

Sinn und Zweck, der über das Drop-down-Menü "Profil/Rolle" P04 auswählbaren Profile ist es, die XML-Elemente zur Ver-/Entschlüsselung für absehbare Anwendungsszenarien vorzugeben.

Sollte augenscheinlich kein Profil zur Nutzerrolle/Aufgabe passen, so wählt man das Profil "Individuell": Es erlaubt als einziges Profil die freie Auswahl der zu ver-/entschlüsselnden XML-Elemente.



Dieser Expertenmodus ist nur für erfahrene Anwender geeignet.

Das folgende Beispiel in Abbildung 6 zeigt die Auswahl der verfügbaren Elemente.

Abbildung 6: Individuelle Auswahl zu verschlüsselnder Elemente

Elemente und öffentliche Schlüssel	
<input checked="" type="checkbox"/> qs_data	PublicKey1
<input checked="" type="checkbox"/> care_provider	PublicKey1
<input checked="" type="checkbox"/> patient	PublicKey2
<input checked="" type="checkbox"/> patient_tx	PublicKey2

### Verschlüsseln von Zertifikatsdateien (.cer) im P-Mode

Eine Zertifikatsdatei mit Endung ".cer" verschlüsselt man im P-Mode wie folgt:

1. Auswahl der zu verschlüsselnden (.cer)-Datei über Knopf P01.
2. Im drop-down-Menü "Aktion" (P03) ist nun "Verschlüsseln", im Drop-down-Menü "Profil/Rolle" (P04) ist die Rolle "PSP Zertifikatstausch" vorausgewählt und die Checkbox "Transportverschlüsselung" ist aktiviert.
3. Man gibt nun das für die Verschlüsselung benötigte Passwort in Feld "Passwort" (P07) ein.

4. Über Knopf P10 wählt man den Ausgabeordner aus, in dem die verschlüsselte Datei ausgegeben werden soll.
5. Feld P11 zeigt den Pfad des ausgewählten Ausgabeordners an.
6. Man klickt auf den Knopf "Ausführen" (P12).
7. Als Ergebnis sollte man eine komprimierte und verschlüsselte Datei im erzeugten Unterordner des Ausgabeordners vorfinden, deren Name das Format T-Zertifikat-<Zeitstempel>.cer.zip.aes besitzt.

### **Entschlüsseln von Zertifikatsdateien (.cer) im P-Mode**

Eine im P-Mode verschlüsselte Datei Zertifikatsdatei mit Endung ".cer.zip.aes" entschlüsselt man wie folgt:

1. Auswahl der zu entschlüsselnden (.cer.zip.aes) Datei
2. Im drop-down-Menü "Aktion" (P03) ist nun "Verschlüsseln", im Drop-down-Menü "Profil/Rolle" (P04) ist die Rolle "PSP Zertifikatstausch" vorausgewählt und die Checkbox "Transportverschlüsselung" ist aktiviert.
3. Man gibt nun das für die Entschlüsselung benötigte Passwort in Feld "Passwort" (P07) ein.
4. Über Knopf P10 wählt man den Ausgabeordner aus, in dem die entschlüsselte Datei ausgegeben werden soll.
5. Feld P11 zeigt nun den Pfad des ausgewählten Ausgabeordners an.
6. Man klickt auf den Knopf "Ausführen" (P12).
7. Als Ergebnis sollte man eine entschlüsselte Zertifikatsdatei im erzeugten Unterordner des Ausgabeordners vorfinden.

### **Verschlüsseln von XML-Dateien im P-Mode in der Rolle "Datenflussprotokoll Strukturabfragen"**

1. Auswahl der zu verschlüsselnden XML-Datei über Knopf P01.
2. Im Drop-down-Menü "Aktion" (P03) wählt man "Verschlüsseln".
3. Im Drop-down-Menü "Profil/Rolle" (P04) wählt man nun die Rolle "Datenflussprotokoll Strukturabfragen" aus.
4. In Feld "Passwort" (P07) gibt man das Passwort ein, mit dem die unter 1. ausgewählte Datei verschlüsselt werden soll.
5. Über Knopf P10 wählt man den Ausgabeordner aus, in dem die verschlüsselte Datei ausgegeben werden soll.
6. Man klickt auf den Knopf "Ausführen" (P12).
7. Als Ergebnis sollte man eine komprimierte und verschlüsselte Datei im erzeugten Unterordner des Ausgabeordners vorfinden, deren Name das Format T-<Zeitstempel>.zip.aes besitzt.

## Entschlüsseln von XML-Dateien im P-Mode in der Rolle "Datenflussprotokoll Strukturabfragen"

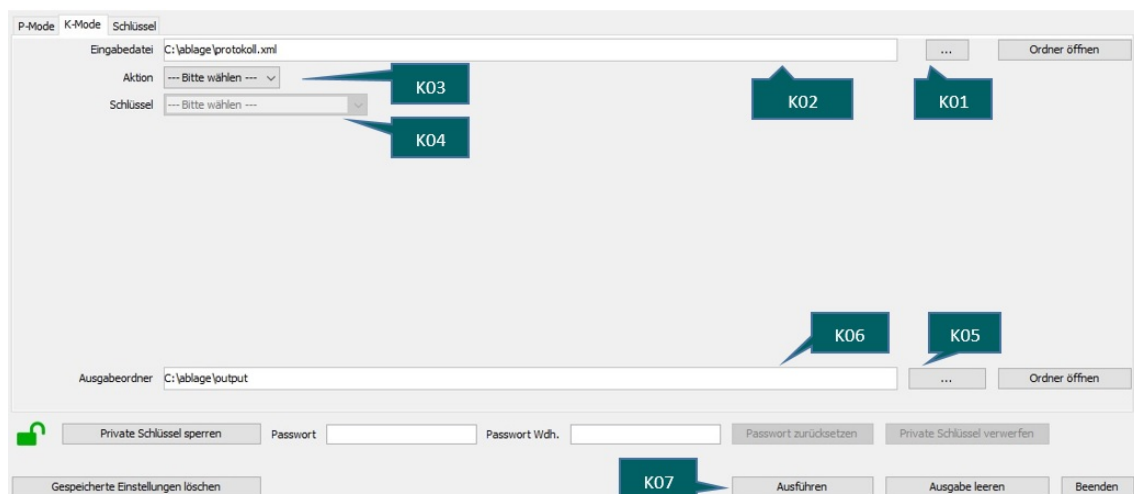
1. Auswahl der zu entschlüsselnden ".zip.aes"-Datei über Knopf P01.
2. In Drop-down-Menü "Aktion" (P03) ist nun "Entschlüsseln" ausgewählt.
3. Im Drop-down-Menü "Profil/Rolle" (P04) wählt man nun die Rolle "Datenflussprotokoll Strukturabfragen" aus.
4. In Feld "Passwort" (P07) gibt man das Passwort ein, mit dem die ursprüngliche XML-Datei entschlüsselt worden ist.
5. Über Knopf P10 wählt man den Ausgabeordner aus, in dem die entschlüsselte Datei ausgegeben werden soll.
6. Man klickt auf den Knopf "Ausführen" (P12).
7. Als Ergebnis sollte man die ursprüngliche entschlüsselte XML-Datei im erzeugten Unterordner des Ausgabeordners vorfinden.

## 8.3 K-Mode

Der K-Mode ermöglicht es, XML-Dateien oder Zertifikatsdateien (.cer) hybrid mit einem öffentlichen Schlüssel zu verschlüsseln. Dies bedeutet insbesondere, dass kein Passwort zur Ver- bzw. Entschlüsselung benötigt wird. Stattdessen kommen die öffentlichen und privaten Schlüssel, die über den Schlüssel-Tab verwaltet werden, zum Einsatz. Verschlüsselt wird die Datei als Ganzes. Es handelt sich also um eine Transportverschlüsselung.

### 8.3.1 Bedienelemente

Abbildung 7: Bedienelemente K-Mode



1. **K01** Knopf "...": Über diesen Knopf kann die zu ver- bzw. entschlüsselnde Datei ausgewählt werden.

2. **K02** Feld "Eingabedatei": Nach der Auswahl über K01 wird hier der Pfad der zu ver- bzw. entschlüsselnden Datei angezeigt.
3. **K03** Drop-down-Menü "Aktion": Über dieses Drop-down-Menü kann angegeben werden, ob die Eingabedatei ver- oder entschlüsselt werden soll.
4. **K04** Drop-down-Menü "Schlüssel": Über dieses Drop-down-Menü kann aus der öffentlichen bzw. privaten Schlüsselliste der Schlüssel ausgewählt werden, der für die Ver- bzw. Entschlüsselung verwendet werden soll.
5. **K05** Knopf "...": Über diesen Knopf kann der Ausgabeordner ausgewählt werden, in dem die aus der Verarbeitung resultierende Datei abgelegt wird.
6. **K06** Feld "Ausgabeordner": Zeigt den Pfad des Ausgabeordners an, der über K05 ausgewählt worden ist.
7. **K07** Knopf "Ausführen": Stösst die Ent- bzw. Verschlüsselung an.

### 8.3.2 Verschlüsseln und Entschlüsseln von Dateien im K-Mode

#### Verschlüsseln von XML-Dateien im K-Mode

Voraussetzung ist, dass man zuvor über die Schlüsselverwaltung den öffentlichen Schlüssel, mit dem man verschlüsseln möchte, bereits hinzugefügt hat. Um eine XML-Datei im K-Mode zu verschlüsseln, geht man wie folgt vor:

1. Über Knopf K01 wählt man die zu verschlüsselnde XML-Datei aus.
2. Der Pfad der ausgewählten Datei erscheint dann in Feld "Eingabedatei" (K02).
3. Im Drop-down-Menü "Aktion" (K03) wählt man "Verschlüsseln" aus.
4. In Drop-down-Menü "Schlüssel" (K04) wählt man den Namen des öffentlichen Schlüssels aus, mit dem man die Datei verschlüsseln möchte.
5. Über den Knopf K05 wählt man den Ausgabeordner aus, in dem die Ausgabedatei abgelegt werden soll.
6. In Feld "Ausgabeordner" (K06) steht nun der Pfad des ausgewählten Ausgabeordners.
7. Nun klickt man auf Knopf "Ausführen" (K07).
8. Als Ergebnis erhält man eine komprimierte und verschlüsselte Datei, deren Namen das Format T-<Zeitstempel>.zip.aes besitzt.

#### Entschlüsseln von .zip.aes-Dateien im K-Mode

Auch hier muss zuvor über die Schlüsselverwaltung der private Schlüssel hinzugefügt worden sein, mit dem entschlüsselt werden soll und die private Schlüsselliste muss entsperrt sein. Um eine vom G-Packer im K-Mode erzeugte ".zip.aes"-Datei zu entschlüsseln, geht man wie folgt vor:

1. Über Knopf K01 wählt man die zu entschlüsselnde .zip.aes-Datei aus.
2. Der Pfad der ausgewählten Datei erscheint dann in Feld "Eingabedatei" (K02).
3. In Drop-down-Menü "Aktion" (K03) ist "Entschlüsseln" bereits vorausgewählt
4. In Drop-down-Menü "Schlüssel" (K04) wählt man den Namen des privaten Schlüssels aus, mit dem man die Datei verschlüsseln möchte.
5. Über den Knopf K05 wählt man den Ausgabeordner aus, in dem die Ausgabedatei abgelegt werden soll.
6. In Feld "Ausgabeordner" (K06) steht nun der Pfad des ausgewählten Ausgabeordners.
7. Nun klickt man auf Knopf "Ausführen" (K07).
8. Als Ergebnis erhält man die entschlüsselte und entpackte Datei mit ihrem ursprünglichen Namen.

### **Verschlüsseln von Zertifikatsdateien (.cer) im K-Mode**

Die Verschlüsselung von Zertifikatsdateien erfolgt analog zum "Verschlüsseln von XML-Dateien im K-Mode" mit dem Unterschied, dass nach Auswahl der (.cer)-Datei über K01 als Aktion bereits "Verschlüsseln" vorausgewählt wird. Der Name der resultierenden Datei hat das Format T-Zertifikat-<Zeitstempel>.cer.zip.aes.

### **Entschlüsseln von verschlüsselten Zertifikatsdateien (.cer.zip.aes) im K-Mode**

Um eine vom GPacker erzeugte (.cer.zip.aes)-Datei zu entschlüsseln, geht man analog zu "Entschlüsseln von .zip.aes-Dateien im K-Mode" vor.

## 8.4 Fehlermeldungen

### Fehlermeldungen GPacker allgemein

Meldung	Erläuterung
Es konnten keine temporären Dateien erstellt werden.	Es wurde kein Ordner zur Erzeugung temporärer Dateien gefunden.
Die Eingabedatei konnte nicht gefunden werden.	Der GPacker konnte die Eingabedatei nicht finden oder nicht öffnen.

Tabelle 1 Fehlermeldungen des GPackers allgemein (Auswahl)

### Fehlermeldungen GPacker-Schlüsselverwaltung

Meldung	Erläuterung
Der Schlüssel ist ungültig. Fehlerhafte oder falsche Schlüsseldatei	Dieser Fehler tritt auf, wenn die Schlüsseldatei keinen gültigen Schlüssel enthält. Evtl. wurde auch eine private Schlüsseldatei mit einer öffentlichen verwechselt.
Der Schlüsselname darf nicht leer sein	Beim Hinzufügen eines Schlüssels muss dem Schlüssel über Feld S12 ein Namen gegeben werden, nur so kann er dann im K-Mode und im P-Mode in den entsprechenden Dropdown-Menüs (K04 bzw. P09) verwendet werden.

Tabelle 2 Fehlermeldungen der GPacker-Schlüsselverwaltung (Auswahl)

### Fehlermeldungen GPacker K-Mode

Meldung	Erläuterung
Es trat folgender Crypto-Fehler auf: Die gegebene Datei ist keine Datei, die im K-Mode verschlüsselt worden ist. Sie kann daher nicht im K-Mode entschlüsselt werden.	Dieser Fehler tritt auf, wenn man versucht eine (.zip.aes)-Datei zu entschlüsseln, die nicht mit dem GPacker oder TPacker per KMode verschlüsselt wurde, sondern z.B. mit dem GPacker symmetrisch im P-Mode. Sinn dieser Fehlermeldung ist der Hinweis darauf, dass diese Datei nicht im GPacker-K-Mode entschlüsselt werden kann.

Meldung	Erläuterung
Es trat folgender Crypto-Fehler auf: Fehler bei der Entschlüsselung. Es handelt sich evtl. nicht um den richtigen private key.	Ursache des Fehlers ist, dass die zu entschlüsselte Datei mit einem öffentlichen Schlüssel verschlüsselt wurde, die nicht zu dem privaten Schlüssel passt, mit dem beim Auftreten des Fehlers versucht wird zu entschlüsseln.

Tabelle 3 Fehlermeldungen der GPacker-K-Mode (Auswahl)

## Fehlermeldungen GPacker P-Mode

Meldung	Erläuterung
Bitte Registriernummer angeben.	Es muss bei dieser Aktion eine Registriernummer angegeben werden. Dieses wurde Ihnen bei der Registrierung mitgeteilt.
Bitte Passwort angeben.	Es muss bei dieser Aktion ein Transport-Passwort angegeben werden. Dieses wurde Ihnen bei der Registrierung mitgeteilt.

Tabelle 4 Fehlermeldungen des GPackers P-Mode (Auswahl)

## 9 Anwendungsbeispiele

Die folgenden Anwendungsbeispiele sollen häufige Szenarien aus dem Alltag der Beteiligten im Datenfluss wiedergeben. Die Fälle sind gruppiert nach der Rolle des Anwenders.

### 9.1 Leistungserbringer (Kliniken/Praxen)

- Die Datenerfassung in der Dokumentationssoftware ist abgeschlossen und die Exportdatei wurde erstellt und soll nun an die Datenannahmestelle verschickt werden.
- Im GParser muss die XML-Verschlüsselung entsprechend des Datenflusses konfiguriert werden:

Bereich	Verfahren	Datenart	XML-Elemente	Öffentlicher Schlüssel der
Krankenhaus	direkt	QS-Daten	qs_data	BAS
		QS-Daten	qs_data	DAS (LKG)
	Follow-up	QS-Daten	qs_data	DAS (LKG)
		PID-Daten	patient	VST-PSN
		PID-TX-Register	patient_tx	VST-TX
kollektivvertraglich	Follow-up	QS-Daten	qs_data	DAS (KV)
		PID-Daten	patient	VST-PSN
selektivvertraglich	Follow-up	QS-Daten	qs_data	BAS
		PID-Daten	patient	VST-PSN

Tabelle 5 Hybride Verschlüsselung der XML-Elemente gemäß DeQS-RL

Bereich	Datenart	XML-Elemente	Öffentlicher Schlüssel der
kollektivvertraglich	PB-Daten	qs_data	BAS
	PID-Daten	patient	VST

Hybride Verschlüsselung der XML-Elemente gemäß oKFE-RL

#### Beispiel 1: Anwender im stationären Sektor

Der Anwender ist im stationären Sektor tätig. Er muss die Elemente qs\_data und patient auswählen und diese mit dem öffentlichen Schlüssel der Datenannahmestelle (qs\_data) und der Vertrauensstelle (patient) verschlüsseln. Falls eine Datenlieferung für das Verfahren QS TX für den Versand vorbereitet werden soll, ist zwingend auch die Verschlüsselung des Elementes patient\_tx erforderlich.

#### Beispiel 2: Anwender im ambulanten Sektor (kollektivvertraglich), Verfahren gem. DeQS-RL

Der Anwender ist im ambulanten Sektor tätig. Er muss die Elemente qs\_data und patient auswählen und diese mit den öffentlichen Schlüsseln der zuständigen Datenannahmestelle (qs\_data) und der Vertrauensstelle (patient) verschlüsseln.

**Beispiel 3: Anwender im ambulanten Sektor (selektivvertraglich), Verfahren gem. oKFE-RL**

Der Anwender ist im ambulanten Sektor tätig. Er muss die Elemente qs\_data und patient auswählen und diese mit den öffentlichen Schlüsseln der Bundesauswertungsstelle (qs\_data) und der Vertrauensstelle (patient) verschlüsseln.

**Beispiel 4: Anwender im ambulanten Sektor, Verfahren gem. oKFE-RL**

Der Anwender ist im ambulanten Sektor tätig. Er muss die Elemente qs\_data und patient auswählen und diese mit den öffentlichen Schlüsseln der Bundesauswertungsstelle (qs\_data) und der Vertrauensstelle (patient) verschlüsseln.

**Beispiel 5: Versendung QS-Sollstatistik**

Die QS-Sollstatistik wurde mit der Dokumentationssoftware erstellt und soll nun an die Datenannahmestelle verschickt werden.

**Beispiel 6: Entschlüsselung Empfangsbestätigung/Datenflussprotokoll**

Die Empfangsbestätigung oder das Datenflussprotokoll wurde zur Verfügung gestellt und muss nun entpackt werden.

## 9.2 Datenannahmestellen

**Beispiel 7: Lieferung im stationären Sektor bearbeiten**

QS-Daten müssen entschlüsselt werden.

**Beispiel 8: Lieferung im ambulanten Sektor (kollektivvertraglich), Verfahren gem. DeQS-RL**

QS-Daten müssen entschlüsselt werden.

**Beispiel 9: Lieferung im ambulanten Sektor (selektivvertraglich), Verfahren gem. DeQS-RL**

Keine Entschlüsselung von QS-Daten.

**Beispiel 10: Lieferung im ambulanten Sektor (kollektivvertraglich), Verfahren gem. oKFE-RL**

Keine Entschlüsselung von PB-Daten.

**Beispiel 11: Weiterleitung an VST (stationär)**

QS-Daten aus dem stationären Sektor sowie pseudonymisierte LE-Kennungen zur Weiterleitung an die Vertrauensstelle verschlüsseln und verpacken.

**Beispiel 12: Weiterleitung an VST (ambulant, kollektivvertraglich), Verfahren gem. DeQS-RL**

QS-Daten aus dem ambulanten Sektor sowie pseudonymisierte LE-Kennungen zur Weiterleitung an die Vertrauensstelle verschlüsseln und verpacken.

**Beispiel 13: Weiterleitung an VST (ambulant, selektivvertraglich), Verfahren gem. DeQS-RL**

Pseudonymisierte LE-Kennungen zur Weiterleitung an die Vertrauensstelle verschlüsseln und verpacken.

**Beispiel 14: Weiterleitung an VST (ambulant, kollektivvertraglich), Verfahren gem. oKFE-RL**  
Pseudonymisierte LE-Kennungen zur Weiterleitung an die Vertrauensstelle verschlüsseln und verpacken.

**Beispiel 15: Lieferung Sollstatistik**

Entschlüsselung der Sollstatistik.

**Beispiel 16: Weiterleitung Sollstatistik an das IQTIG**

Verschlüsselung der Sollstatistik zur Weiterleitung an das IQTIG.

**Beispiel 17: Datenflussprotokoll des IQTIG entschlüsseln**

**Beispiel 18: Datenflussprotokoll des IQTIG für den LE verschlüsseln**

**Beispiel 19: Empfangsbestätigung der VST entschlüsseln**

# 10 Technische Spezifikation

## 10.1 Hintergrund der K-Mode-Verschlüsselung

Bei der K-Mode Verschlüsselung handelt es sich um eine hybride Verschlüsselung. Die zu verschlüsselnde Datei wird zunächst mit einem zufällig erzeugten 128-bit-Schlüssel AES-verschlüsselt (symmetrisches Verfahren). Dieser SecretKey (oder auch Sitzungsschlüssel) wird anschließend RSA-verschlüsselt (asymmetrisches Verfahren). Einerseits können durch die Verwendung des AES-Verfahrens längere Nachrichten verschlüsselt werden, als es möglich wäre, wenn man auf ein reines RSA-Verfahren setzen würde, und andererseits benötigt man durch den Einsatz der asymmetrischen RSA-Verschlüsselung des symmetrischen Schlüssels (SecretKey) keinen sicheren Kanal, um beispielsweise ein Passwort auszutauschen. Um eine Datei im K-Mode zu verschlüsseln, benötigt man lediglich den öffentlichen Schlüssel des Empfängers. Der Empfänger kann dann mit seinem RSA PrivateKey den SecretKey entschlüsseln und mit diesem SecretKey dann die eigentliche Payload, die vom Sender AES-verschlüsselt worden ist, wieder entschlüsseln. Es wird jetzt schon deutlich, dass der Sender neben der eigentlichen verschlüsselten Payload auch den verschlüsselten SecretKey mitschicken muss. Es werden sogar noch mehr Informationen benötigt. Diese alle werden aneinandergehängt und als Datei mit einem bestimmten Transportformat dem Sender übermittelt. Dieses Transportformat wird genauer in Abschnitt D spezifiziert. Die folgenden Abschnitte enthalten die genauere Spezifikation des eingesetzten AES-Verfahrens sowie des RSA-Verfahrens.

## 10.2 Spezifikation der K-Mode-Verschlüsselung

### A Details zum Verfahren insgesamt

1. Hybrides Verfahren: Die zu verschlüsselnde Datei wird zunächst mit einem zufällig erzeugten 128-bit-Schlüssel AES-verschlüsselt (symmetrisches Verfahren), anschließend wird dieser Schlüssel RSA-verschlüsselt (asymmetrisches Verfahren).
2. Die AES-verschlüsselte Payload wird zusammen mit dem RSA-verschlüsselten Schlüssel und anderen Informationen zu einer Datei zusammengefügt (Siehe Abschnitt D).

### B Details zum verwendeten AES-Verfahren

- Cipher Type: AES/GCM/Nopadding (GCM steht für "Galois Counter Mode")
- Schlüssellänge: 128 bit
- GCM-Tag-Länge: 128 (Authentifizierungstag)

- Länge des Initialisierungsvektors: 12 byte

## C Details zum verwendeten RSA-Verfahren

- Cipher Type: RSA/ECB/OAEPwithSHA1andMGF1Padding

## D Aufbau des Transportformates einer im K-Mode verschlüsselten Datei

Das Transportformat einer im K-Mode verschlüsselten Datei besteht nicht nur aus dem eigentlichen verschlüsselten Inhalt einer Datei, sondern zusätzlich auch einer Reihe von Informationen, die zur Entschlüsselung benötigt werden. Diese Informationen werden alle aneinandergehängt und als eine Datei ausgegeben. Dieser Abschnitt beschreibt den genauen Aufbau dieses Transportformates. Die folgende Tabelle enthält pro Zeile einen definierten Abschnitt unterschiedlicher Länge. Um die Datei insgesamt zu erzeugen, müssen die einzelnen in der Tabelle aufgelisteten Abschnitte aneinandergehängt werden.

Abschnitt	Anzahl bytes	Beschreibung
1	9	Der String "KMODEv001", kodiert in UTF-8, also in Hexadezimal-Schreibweise 4B 4D 4F 44 45 76 30 30 31 (Magic Bytes). Dient als Erkennungsmerkmal einer im K-Mode verschlüsselten Datei.
2	4	Die Länge des RSA-verschlüsselten Schlüssels in Anzahl der Bytes in binärer Zahlendarstellung mit den höherwertigen bytes zuerst. (Gemeint ist die Länge des symmetrischen Schlüssels nach der RSA-Verschlüsselung. Dieser wird in Abschnitt 3 abgelegt.) Diese Länge hängt vor allem von der Länge des verwendeten RSA-Schlüssels ab.
3	Anzahl, die durch Abschnitt 2 definiert wird.	Der RSA-verschlüsselte symmetrische Schlüssel
4	12	Initialisierungsvektor für den Galois Counter Mode der AES-Verschlüsselung.
5	je nach Datenmenge	Die AES-verschlüsselten Daten. (Wurden mit dem Schlüssel verschlüsselt, der unter Abschnitt 3 RSA-verschlüsselt abgelegt worden ist.)

*Aufbau einer K-Mode-verschlüsselten Datei*

In der folgenden Abbildung sieht man ein Beispiel einer K-Mode-verschlüsselten Datei, geöffnet in einem Hexeditor.

- Rot unterstrichen sieht kommt zuerst der kodierte "KMODEv001"-String

- Grün unterstrichen sieht man in den bytes 00 00 01 00 die Zahl 256 kodiert. Sie beschreibt die Länge des im folgenden Abschnitt abgelegten Schlüssels.
- Blau unterstrichen folgen die 256 bytes des RSA-verschlüsselten symmetrischen Schlüssel
- Gelb unterstrichen ist der 12-byte lange Initialisierungsvektor
- Grau unterstrichen folgen die verschlüsselten Daten. Die graue Linie endet mit drei grauen Punkten, um anzudeuten, dass es sich um den gesamten Rest der Datei handelt.

```

T-2023_06_26_170908.zip.ape
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000 BB 4D 4F 44 45 76 30 30 31 09 0A 00 0C 0D 56 0F 13 3E 27 F6 81 60 03 9B EF 40 63 45 79 58 08 D6
00000020 FF 8D 4A 6A 71 C4 77 E1 6D 87 05 1E 0B B4 F1 EA AA 9B DC 44 B8 B4 BF 31 DB 54 17 6A BC 77 2D 0B
00000040 53 7E 40 98 52 11 50 B5 40 47 0E B1 29 25 80 B1 6A 87 0B EA 3A BA ED 45 04 72 91 D6 95 08 D6
00000060 74 6B C4 25 3C 51 76 3A B6 57 15 04 0D E3 CE 6E 51 0E 39 CE 01 00 28 0E 1E 2C 56 3F 0E
00000080 74 6B C4 25 3C 51 76 3A B6 57 15 04 0D E3 CE 6E 51 0E 39 CE 01 00 28 0E 1E 2C 56 3F 0E
000000A0 2E 1E 18 41 0D 2B 83 2C 5F 49 5A 58 D5 91 92 87 0F 9A 7D 84 2A 9A 80 83 43 2B 1D 1D BB 04
000000C0 6D EA F3 2E 21 0C 83 6F 4D 90 95 3F 25 28 3C 64 2D 8D 77 F5 06 44 1E 9E 42 E7 73 D4 04
000000E0 6D EA F3 2E 21 0C 83 6F 4D 90 95 3F 25 28 3C 64 2D 8D 77 F5 06 44 1E 9E 42 E7 73 D4 04
00000100 08 5E 16 BE A6 9E 28 D2 B6 04 B2 17 13 30 15 EE DA 79 04 79 2C 8B 35 C2 BF B0 71 C0 74 70 4B
00000120 BB 35 EA FA 51 BC 36 3F 25 BB 0A F9 80 0B B8 D5 93 F0 8C 40 40 95 3C C2 76 F7 B1 C1 52 79 2D
00000140 A6 24 56 A5 98 B5 D5 12 D3 20 25 56 50 83 BC 79 BA DA BC DE 13 5A AF F4 89 7E 6C 4B BA 99 D3 B9
00000160 D3 14 1E 0F A1 07 1B 0A CE 1D 06 EF 6E 1F 6F EA 3D D2 F2 51 22 74 85 A1 48 CF CF 06 06 87 73 10
00000180 9B 3C C8 C9 F0 02 B2 7D E5 21 2C EF E1 6E 9E 27 3C EF DD 15 1C 78 73 67 D7 AE D6 57 91 E3 AD
000001A0 A8 3B DA EF BB 6E 70 29 24 99 FF C0 D2 85 39 03 F1 3D 9F AF DF AB 65 48 0B 08 71 D1 3D 2F D2
000001C0 6E A0 EF ED BA 9B AC 33 1E 3F 3B C1 4C 61 3C 2A EA 01 A8 2C 9A 31 96 AD 17 93 99 61 DF CF C9
000001E0 EC DC BB C9 97 D5 D1 6A 68 7D A8 54 95 4D C9 BD 68 61 64 17 6B 5F 6B 85 B3 F5 04 2B C1 BA A6
00000200 C1 97 B9 DA 20 72 B7 D0 A3 C7 BB 5F 5F DE 05 B6 79 6D 6B 2F 8E 8F 94 81 90 9D 8E 22 1C 25 4F
00000220 29 E9 CF 37 F3 0E EC A4 C9 24 EA 20 08 EB 2D E4 CC 00 EA 44 42 67 94 CC A3 70 3B B1 68 44 E2 50
00000240 D6 74 1E 57 37 13 30 10 E6 47 C2 F2 C6 5F 4A AE B6 C3 71 42 41 70 99 7D F5 CD 1A 3F 51 02 19 41 D4 45
00000260 55 AE F3 2D 8A 6C 54 F9 33 7A 2B ED 01 04 DE B3 C9 EE 0A B7 30 44 DA 07 F2 44 1F DE EA 11 BC 4E
00000280 9D F2 0E 02 A6 B5 BF D4 54 D4 D9 0C D4 81 78 7E 14 47 70 99 7D F5 CD 1A 3F 51 02 19 41 D4 45
000002A0 41 2B 53 9F A6 45 37 86 53 1F BF 8B 0B 3B CA D0 82 D0 68 A3 24 CE 03 1D EB 07 D7 F9 93 5D 90 3E
000002C0 F4 47 D5 FD 18 55 A6 02 D4 07 27 4A 5F 5F DE 05 B6 79 6D 6B 2F 8E 8F 94 81 90 9D 8E 22 1C 25 4F
000002E0 8C DD 36 2D 84 C8 95 9B 34 80 B6 64 0C 0E 55 51 0E F9 F6 1D 08 8B A7 22 B6 E2 EA 9F CE 46 AB 87
00000300 ED EA 51 E4 45 44 29 19 B7 7B FE AD EE 4B 7A 88 A7 64 90 CD 2A 9A 51 D2 F6 11 74 FB 28 35 49 DA
00000320 B7 8D D0 1A 5E 96 32 A4 C3 2C B8 0B 08 53 94 46 3F 80 D0 BB 7E 4A 7B 0A 76 CA 3D 3F DE 53 09
00000340 15 71 A4 78 92 22 6C CB 2A A6 BD A5 05 5B 7C 74 6C C3 36 D7 9A 22 41 AF EA 8C 86 11 5F 3B BD 12
00000360 74 0A 84 28 03 1F 3B 6D C8 4F 14 BD 30 8B AB CA 6E 8A 9A 06 4F D2 D1 F4 99 CF 8E AD 42 83 57
00000380 10 C4 25 85 D5 9E 85 61 51 21 40 DD 20 93 B8 B1 F7 6D 5A 6A B0 EA 93 14 D6 65 88 EA D2 75 29 96
000003A0 B5 91 D5 87 DE 59 D0 1F 0A E7 4F 91 DC 2F 5E E8 D0 23 CD 4D CA 2A 89 D0 2C 70 5B D8 9D 08 B1 B8
000003C0 80 AD 87 A5 11 A5 1C 1E 1A 3A FF E3 25 A8 18 29 84 C9 44 FA EC 0C 68 68 5A A2 1D ED CC 01
000003E0 16 94 7D 88 83 8C 01 85 D3 07 07 CE 3F EF B9 18 72 5E 52 9B AE EA D CBF 6A D9 D2 79 03 BA 5F
00000400 8C D9 84 38 A6 40 A0 AE A7 3F 80 B7 74 F1 68 61 EF F2 52 2F 5F 19 B3 51 7C 3B BD 27 AD
00000420 AC 59 D1 E4 92 67 16 4B 8B BF 63 4E 9C 13 92 08 03 6D D4 D4 A7 5D DF CF 07 15 85 DF 29 1F 74
00000440 DF 30 5E 67 B0 F4 87 89 9E 88 E8 5E 7C 79 3D 13 CF B1 6F 01 E4 03 A0 C7 CC 72 65 CE F6

```

## E Kompatibilität mit den TPACK- und GPACK-Anwendungen

Der TPacker entschlüsselt dagegen auch Dateien, die im K-Mode verschlüsselt aber vorher nicht gezippt worden sind. Insgesamt empfiehlt es sich als Best-Practice die im K-Mode zu verschlüsselende Datei vorher zu zippen und dann im K-Mode zu verschlüsseln. Erstens um die Dateigröße zu minimieren und zweitens um ein Maximum an Kompatibilität zu gewährleisten.

- Bei der K-Mode Verschlüsselung sollte man unbedingt sowohl den erzeugten AES-Schlüssel als auch den Initialisierungsvektor aus einer guten Zufallsquelle zu beziehen.

- Sowohl der AES-Schlüssel als auch der Initialisierungsvektor sollten unbedingt für jede zu verschlüsselnde Datei neu erzeugt werden und keinesfalls wiederverwendet werden.

## G Code-Beispiele

Beispiel 1: Java-Methode, die Datei in entsprechendem Transportformat erzeugt:

```
...
private static final String AES_GCM_OPERATION_MODE = "AES/GCM/NoPadding";
...
public static final String FILE_TYPE_IDENTIFIER_NEW = "KMODEv001";
...
/**
 * Verschlüsselt den Inhalt eines eingehenden InputStreams und
 * schreibt das Resultat auf den gegebenen OutputStream.
 * Es wird eine symmetrische Verschlüsselung (AES) durchgeführt. Als
 * Betriebsmodus wird GCM (Galois Counter Mode) verwendet.
 * <p>
 * Vor das Chifftrat werden das Salt und die IV gehangen. In
 * alternativen Implementierungen könnte man über eine Base64 Kodierung des
 * Salt und der
 * IV nachdenken.
 * Das ist hier aber unnötig.
 * Salt+IV+Cipher
 * <p>
 * Zum Entschlüsseln müssen zuerst das Salt und die IV gelesen
 * werden.
 *
 * @param inputStream die zu verschlüsselnden Daten als InputStream
 * @param outputStream der OutputStream auf den das
 * Verschlüsselungsergebnis geschrieben werden soll
 * @param secretKey der {@link SecretKey} der zur Anwendung kommt
 * @param secretKeyWrapper eine Function, die das Wrapping des
 * SecretKeys umsetzt
 */
public void encrypt(final InputStream inputStream, final OutputStream
outputStream, final SecretKey secretKey, final Function<SecretKey,
byte[]> secretKeyWrapper) throws CryptoException {

    try {
        // https://en.wikipedia.org/wiki/Galois/Counter_Mode
        // GCM Tag Length kann einer der folgenden Werte sein 128,
        120, 112, 104, or 96.
        // Wir nehmen einfach 128
        final IvParameterSpec ivParameterSpec = createRandomIV();
```

```

        final GCMParameterSpec gcmParameterSpec = new
GCMParameterSpec(128, ivParameterSpec.getIV());
        final Cipher cipher =
Cipher.getInstance(AES_GCM_OPERATION_MODE);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey,
gcmParameterSpec);

        try (final CipherOutputStream cipherOutputStream = new
CipherOutputStream(outputStream, cipher)) {

            // Schreibe MagicBytes um den Dateityp erkennbar zu
machen

outputStream.write(FILE_TYPE_IDENTIFIER_NEW.getBytes(StandardCharsets.UTF
_8));

            // Der write der IV muss auf dem outputStream und nicht
auf dem cipherOutputStream erfolgen, weil diese nicht verschlüsselt
werden sollen.
            byte[] wrappedSecretKey =
secretKeyWrapper.apply(secretKey);

            // Schreibe die Länge des SecretKeys
            // Code aus dem DataOutputStream entwendet.
            final int lengthOfWrappedSecretKey =
wrappedSecretKey.length;
            //System.out.println(lengthOfWrappedSecretKey);
            outputStream.write((lengthOfWrappedSecretKey >>> 24) &
0xFF);
            outputStream.write((lengthOfWrappedSecretKey >>> 16) &
0xFF);
            outputStream.write((lengthOfWrappedSecretKey >>> 8) &
0xFF);
            outputStream.write((lengthOfWrappedSecretKey >>> 0) &
0xFF);

            outputStream.write(wrappedSecretKey);
            outputStream.write(ivParameterSpec.getIV());
            transferTo(inputStream, cipherOutputStream);
        } catch (IOException e) {
            throw new RuntimeException("Unhandled exception
occurred.", e);
        }
        } catch (NoSuchAlgorithmException e) {
            throw new CryptoException(new
CryptoError(CryptoErrorId.NO_SUCH_ALGORITHM_ERROR,

```

```

AES_GCM_OPERATION_MODE, e.getMessage()));
    } catch (NoSuchPaddingException |
InvalidAlgorithmParameterException | InvalidKeyException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.CRYPTO_ERROR));
    }
}

```

Beispiel 2: Java-Methode, die Datei in Transportformat entschlüsselt:

```

...
private static final String AES_GCM_OPERATION_MODE = "AES/GCM/NoPadding";
...

/**
 * Entschlüsselt den eigenhenden InputStream und schreibt das
 * Resultat auf den gegebenen OutputStream.
 *
 * @param inputStream die verschlüsselten Daten als InputStream
 * @param outputStream der OutputStream auf den das
 * Entschlüsselungsergebnis geschrieben werden soll
 * @param secretKeyUnwrapper Function die den extrahierten SecretKey
 * unwrapped
 */
public void decrypt(final InputStream inputStream, final OutputStream
outputStream, final Function<byte[], SecretKey> secretKeyUnwrapper)
throws CryptoException {

    final byte[] ivBytes = new byte[12];

    try {
        // man könnte noch überprüfen ob die bytes korrekt gelesen
        wurden. Ist mir aber erstmal egal.
        // Deswegen meckert Sonar in der IDE

        // Lese zuerst die MagicBytes um den Dateityp zu erkennen.
        if
(!EncryptionType.KMODE.equals(checkEncryptionType(inputStream))) {
            throw new CryptoException(new
CryptoError(CryptoErrorId.WRONG_FILE_TYPE));
        }

        // Lese die Länge des SecretKeys
        // Code aus dem DataInputStream entwendet.
        int ch1 = inputStream.read();

```

```

        int ch2 = inputStream.read();
        int ch3 = inputStream.read();
        int ch4 = inputStream.read();
        if ((ch1 | ch2 | ch3 | ch4) < 0) {
            throw new EOFException();
        }
        int lengthOfSecretKeyBytes = ((ch1 << 24) + (ch2 << 16) +
        (ch3 << 8) + (ch4 << 0));

        final byte[] secretKeyBytes = new
byte[lengthOfSecretKeyBytes];
        inputStream.read(secretKeyBytes);
        inputStream.read(ivBytes);
        final SecretKey secretKey =
secretKeyUnwrapper.apply(secretKeyBytes);
        final GCMParameterSpec gcmParameterSpec = new
GCMParameterSpec(128, ivBytes);
        final Cipher cipher =
Cipher.getInstance(AES_GCM_OPERATION_MODE);
        cipher.init(Cipher.DECRYPT_MODE, secretKey,
gcmParameterSpec);

        try (final CipherInputStream cipherInputStream = new
CipherInputStream(inputStream, cipher)) {
            transferTo(cipherInputStream, outputStream);
        } catch (IOException e) {
            throw new RuntimeException("Unhandled exception
occurred.", e);
        }
    } catch (IOException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.IO_ERROR));
    } catch (NoSuchPaddingException |
InvalidAlgorithmParameterException | InvalidKeyException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.CRYPTO_ERROR));
    } catch (NoSuchAlgorithmException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.NO_SUCH_ALGORITHM_ERROR,
AES_GCM_OPERATION_MODE, e.getMessage()));
    }
}

```

Beispiel 3: Methode, die die Magic-Bytes ausliest. (In einer vorherigen Version begannen die Magic-Bytes mit RPACKer..)

```

...
public static final String FILE_TYPE_IDENTIFIER_NEW = "KMODEv001";
...
public EncryptionType checkEncryptionType(InputStream inputStream) {
    try {
        byte[] magicBytesConstant =
FILE_TYPE_IDENTIFIER_NEW.getBytes(FILE_TYPE_IDENTIFIER_CHARSET);
        byte[] magicBytesFile = new byte[magicBytesConstant.length];
        inputStream.read(magicBytesFile);
        String readString = new String(magicBytesFile,
FILE_TYPE_IDENTIFIER_CHARSET);
        boolean rpackerMatch=readString.matches("RPACKERv\\d{1}");
        boolean kmodeMatch=readString.matches("KMODEv\\d{3}");
        if (rpackerMatch||kmodeMatch) {
            return EncryptionType.KMODE;
        } else {
            return EncryptionType.NONKMODE;
        }
    } catch (UnsupportedEncodingException e) {
        throw new RuntimeException("Die interne(!) Zeichenkodierung
'" + FILE_TYPE_IDENTIFIER_CHARSET + "' für die MagicBytes ist ungültig.",
e);
    } catch (IOException e) {
        throw new RuntimeException("Beim Einlesen der Magic-Bytes der
zu entschlüsselnden Datei trat ein IO-Fehler auf.", e);
    }
}
}

```

Beispiel 4: Java-Methode, die den AES-Schlüssel per RSA wrappt (verschlüsselt):

```

...
private static final String CIPHER_TRANSFORMATION =
"RSA/ECB/OAEPwithSHA1andMGF1Padding";
...
public byte[] wrapSecretKey(final SecretKey secretKey, final PublicKey
publicKey) throws CryptoException {
    try {
        // Inspiriert von https://devtut.github.io/java/rsa-encryption.html#an-example-using-a-hybrid-cryptosystem-consisting-of-oaep-and-gcm
        final Cipher cipher =
Cipher.getInstance(CIPHER_TRANSFORMATION);
        cipher.init(Cipher.WRAP_MODE, publicKey);
        return cipher.wrap(secretKey);
    }
}

```

```

        } catch (NoSuchAlgorithmException e) {
            throw new CryptoException(new
CryptoError(CryptoErrorId.NO_SUCH_ALGORITHM_ERROR, CIPHER_TRANSFORMATION,
e.getMessage()));
        } catch (NoSuchPaddingException | InvalidKeyException |
IllegalBlockSizeException e) {
            throw new CryptoException(new
CryptoError(CryptoErrorId.CRYPTO_ERROR));
        }
    }
}

```

Beispiel 5: Java-Methode, die den AES-Schlüssel per RSA unwrappt (entschlüsselt):

```

...
private static final String CIPHER_TRANSFORMATION =
"RSA/ECB/OAEPwithSHA1andMGF1Padding";
...
public SecretKey unwrapSecretKey(final byte[] wrappedSecretKey, final
PrivateKey privateKey) throws CryptoException {
    try {
        final Cipher cipher =
Cipher.getInstance(CIPHER_TRANSFORMATION);
        cipher.init(Cipher.UNWRAP_MODE, privateKey);
        return (SecretKey) cipher.unwrap(wrappedSecretKey,
CryptoConstant.AES.getValue(), Cipher.SECRET_KEY);
    } catch (NoSuchPaddingException | InvalidKeyException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.INVALID_PRIVATE_KEY_ERROR));
    } catch (NoSuchAlgorithmException e) {
        throw new CryptoException(new
CryptoError(CryptoErrorId.NO_SUCH_ALGORITHM_ERROR, CIPHER_TRANSFORMATION,
e.getMessage()));
    }
}
}

```