

# **Asymmetrische Kryptographie in Java**

Norman Vetter

**Seminar „Sichere verteilte Anwendungen mit Java“**

Universität Potsdam

Wintersemester 2012/13

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>3</b>
2.1	Ziele . . . . .	3
2.2	Asymmetrische Kryptographie . . . . .	3
2.3	Grundlage asymmetrischer Systeme . . . . .	4
2.4	RSA . . . . .	5
2.4.1	Schlüsselberechnung . . . . .	5
2.4.2	Verschlüsselung und Entschlüsselung . . . . .	6
2.5	Schlüsselmanagement . . . . .	6
2.6	Hybride Kryptographie . . . . .	7
<b>3</b>	<b>Kryptographie mit Java</b>	<b>8</b>
3.1	Java Cryptography Architecture . . . . .	8
3.2	Zufallszahlen . . . . .	8
3.3	Schlüsselgenerierung . . . . .	9
3.4	Schlüsselspeicherung . . . . .	9
3.5	Schlüsseleinigung . . . . .	9
3.6	Ver- und Entschlüsselung . . . . .	9
<b>4</b>	<b>Zusammenfassung</b>	<b>9</b>

# 1 Einleitung

So sieht eine Section aus!

## 2 Theoretische Grundlagen

### 2.1 Ziele

Ein kryptographisches System dient zum verschlüsseln und entschlüsseln von Texten und anderen Daten, um deren Inhalt vor Dritten geheim zu halten. Genauer gibt ein Kryptosystem an wie ein Klartext in einen von Dritten nicht lesbaren Kryptotext umgewandelt werden kann. Und wie dieser Kryptotext später wieder in einen lesbaren Klartext transformiert wird. Anders als die Steganografie zielt die Kryptographie darauf ab lediglich den Inhalt einer Nachricht zu verschlüsseln, nicht aber deren Existenz zu verbergen. Die asymmetrische Kryptographie ist eines dieser kryptographischen Systeme.

### 2.2 Asymmetrische Kryptographie

Die asymmetrische Kryptographie wurde Mitte 1970 von Ralph Merkle sowie von Diffie und Hellmann entwickelt. Sie beruht auf der Idee zur Kommunikation zwischen 2 Instanzen ein Schlüsselpaar zu verwenden. Dieses Schlüsselpaar besteht aus dem privaten und dem öffentlichen Schlüssel. Zur Kommunikation muss im Vorhinein ein gegenseitiger Austausch des öffentlichen Schlüssels erfolgt sein, denn die zu schickende Nachricht ist vom Sender mit dem öffentlichen Schlüssel des Empfängers in einen Kryptotext um zu wandeln. Nach Empfang entschlüsselt der Empfänger nun die Nachricht mit seinem eigenen privaten Schlüssel. Im Falle einer Antwort verschlüsselt der ehemalige Empfänger (nun Sender) seine Nachricht wieder mit dem öffentlichen Schlüssels seines Gegenüber. Welcher die Entschlüsselung erneut mit dem eigenen privaten Schlüssel durchführen muss. Somit ist während der Kommunikation kein erneuter Austausch eines sicheren Schlüssels notwendig und auch erneute Kommunikationen können mit den bereits vorhandenen Schlüsseln durchgeführt werden. Wichtig ist hierbei jedoch, dass die Sicherheit des privaten Schlüssels und die Authentizität des öffentlichen Schlüssels gewährleistet ist. Mehr dazu in den folgenden Kapiteln.

## 2.3 Grundlage asymmetrischer Systeme

Im obigen Szenario haben wir gesehen wie ein grober Kommunikationsablauf zwischen Zwei Parteien aussieht. Dieses wird in folgender Grafik ( nach [Eck13] ) veranschaulicht:

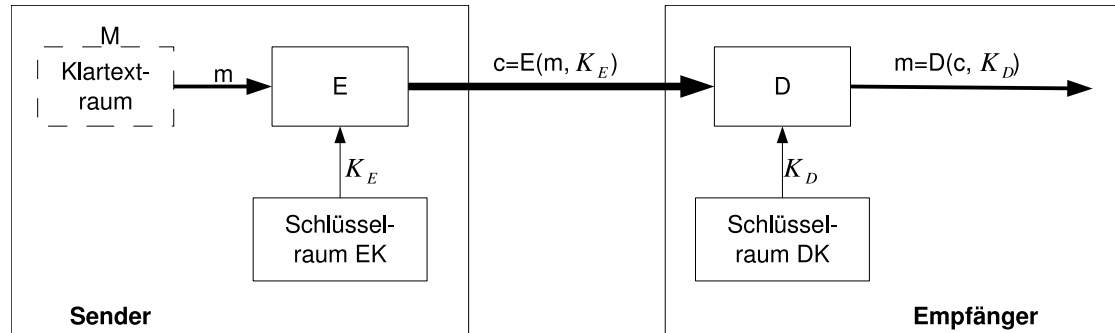


Abbildung 1: Komponenten eines Kryptosystems

- |  |   |
|--|---|
| 1. Tupel<br>( $M, C, EK, DK, E, D$ )                                     | 7. Verschlüsselungsschlüsselraum<br>( $EK \setminus \emptyset$ )  |
| 2. 2 endliche Alphabete<br>( $A_1, A_2$ )                                | 8. Entschlüsselungsschlüsselraum<br>( $DK \setminus \emptyset$ )<br>mit $f : EK \rightarrow DK$<br>und $f(K_E) = K_D$ |
| 3. Menge von Klartexten<br>( $M \subseteq A_1^* \setminus \emptyset$ )   | 9. Verschlüsselungsverfahren<br>( $E : M \times EK \rightarrow C$ )   |
| 4. Klartext<br>( $m \in M$ )   | 10. Entschlüsselungsverfahren<br>( $D : C \times DK \rightarrow M$ )  |
| 5. Menge von Kryptotexten<br>( $C \subseteq A_2^* \setminus \emptyset$ ) | 11. Es gilt:<br>$\forall m \in M : D(E(m, K_E), K_D) = m$   |
| 6. Kryptotext<br>( $c \in C$ )   |   |

Die Eigenschaft (11.) der obigen Legende zeigt uns das Zusammenspiel unserer einzelnen Komponenten. Wird ein Klartext  $m$  mit einem Schlüssel  $K_E \in EK$  durch ein kryptographisches Verfahren  $E$  in einen Kryptotext umgewandelt. So ist gegeben, dass dieser Kryptotext unter Verwendung des zu  $K_E$  gehörigen Schlüssels  $K_D \in DK$ , und des kryptographischen Verfahrens  $D$  wieder in den ursprünglichen Klartext  $m$  umgewandelt werden kann. Damit dies gegeben ist, muss unser asymmetrisches Kryptosystem einige wichtige Punkte erfüllen. Es muss (nach [Eck13] ):

1. eine effiziente Möglichkeit zur Erzeugung von Schlüsselpaaren  $(K_E, K_D)$  geben.
2. garantiert sein, dass der Private  $(K_D)$  nicht effizient aus dem öffentlichen Schlüssel  $(K_E)$  gebildet werden kann.
3. möglich sein effizient zu Ver- und Entschlüsseln.
4. optional aber nicht notwendiger Weise die Möglichkeit bestehen mit beiden Schlüsselpaaren zu verschlüsseln und mit dem jeweils anderem das Resultat zu entschlüsseln. Somit ist das System für Certifikate geeignet.

Um dies zu erreichen nutzen wir Einwegfunktionen.

**Einwegfunktionen** sind besondere Funktionen  $(f : X \rightarrow Y)$  bei denen das Urbild  $(X)$  nicht unter vertretbarem zeitlichem Aufwand aus dem Bild  $(Y)$  zu berechnen ist. Mathematische Probleme, welche derartige Funktionen bilden, sind unter Anderem das Faktorisierungsproblem und der diskrete Algorithmus. Es ist nicht bewiesen, dass die Umkehrung nicht möglich ist, jedoch übersteigt die Komplexität der Berechnung unser heutiges Vermögen diese in einer vertretbaren Zeitspanne zu lösen.

Bei der Verwendung von normalen Einwegfunktionen in einem Kryptosystem wäre zwar die Sicherheit der Daten garantiert, jedoch stellt die Umkehrung selbst für autorisierte Personen ein unüberwindbares Hindernis dar. Wir benötigen zum Entschlüsseln unserer Daten eine Hintertür - die so genannte Falltür. Einwegfunktionen mit Falltür bieten eine identische Sicherheit wie normale Einwegfunktionen, und die Option verschlüsselte Daten unter Kenntnis eines Geheimnisses (bei und der private oder öffentliche Schlüssel) zu entschlüsseln. Beispiele für mathematische Probleme mit Falltür sind die  $h$ -te Potenz modulo  $(n)$  und der zusammengesetzte Modul  $(n)$  ([Eck13]).

## 2.4 RSA

Bei RSA handelt es sich um ein 1978 von Ronald Rivest, Adi Shamir und Leonard Adleman entwickeltem Verfahren ([Eck13]), welches alle Vier von uns genannten Bedingungen für ein asymmetrisches Kryptosystem erfüllt. Es beruht auf dem Faktorisierungsproblem.

### 2.4.1 Schlüsselberechnung

Zur Berechnung unseres Schlüsselpaares mit dem öffentlich Schlüssel  $(e, n)$  und dem privaten Schlüssel  $(d, \varphi(n))$  gehen wir folgenden Weg:

1. wähle zwei zufällige und große Primzahlen  $p \neq q$ .
2. Ermittle  $n = p \cdot q$  und  $\varphi(n) = (p - 1)(q - 1)$ .
3. Wähle eine Zahl  $e$  für die gilt:  $\text{ggT}(e, \varphi(n)) = 1$  und  $1 < e < \varphi(n)$

4. Als nächstes berechne  $d$  mit  $ed = 1 \bmod \varphi(n)$ .
5.  $(n, d)$  ist privater Schlüssel,  $(n, e)$  ist öffentlicher Schlüssel

### 2.4.2 Verschlüsselung und Entschlüsselung

Möchten wir nun einen Klartext in einen Kryptotext umwandeln haben wir zwei Schritte zu befolgen. Als erstes muss der Klartext ( $m$ ) in eine Folge von Zahlen mit  $m_i \in \{0, 1, \dots, n-1\}$  umgewandelt werden. Daraufhin wird für alle  $m_i$   $c_i = m_i^e \bmod n$  mit Hilfe des vorher berechneten öffentlichen Schlüssels bestimmt und wir erhalten den Kryptotext  $c$ . Die Umkehrung erfolgt ähnlich. Jedoch nutzen wir dieses mal den privaten Schlüssel  $(d, n)$ . Zuerst wird der Kryptotext in die Zahlenfolge umgewandelt indem wir  $m_i = c_i^d \bmod n$  für alle  $c_i$  bestimmen. Im Anschluss muss lediglich die Zahlenfolge wieder in unseren Klartext transformiert werden.

Die Möglichkeit dieses Verfahren auch invers an zu wenden, also das Verschlüsseln mit dem privaten und das Entschlüsseln mit dem öffentlichen Schlüssel, wird bei Zertifikaten und E-Mail-Signaturen in Anspruch genommen. Bei letzteren wird aus dem Inhalt der E-Mail ein Hashcode generiert, welcher mit dem privaten Schlüssel verschlüsselt, als Signatur an die Nachricht angehängt und dann verschickt wird. Dabei geht es weniger um das Geheimhalten des Inhaltes der Nachricht. Sondern um das Sicherstellen das sie unverfälscht beim Empfänger eintrifft. Nach Erhalt der E-Mail kann der Empfänger die Signatur mit dem öffentlich Schlüssel des Senders entschlüsseln und den Inhalt der Nachricht mit dem Hashcode abgleichen. Sollte es dabei zu Unstimmigkeiten kommen ist bewiesen das die Nachricht nach anhängen der Signatur verändert wurde.

## 2.5 Schlüsselmanagement

Bei der asymmetrischen Kryptographie ergeben sich aufgrund ihrer besonderen Grundidee der zwei Schlüssel auch ganz eigene Möglichkeiten und Pflichten zur Erzeugung, Aufbewahrung und möglichen Wiederherstellung dieser.

**Die Erzeugung** asymmetrischen Schlüsselmaterials kann der Nutzer entweder selbst übernehmen oder einer zentralen Instanz (z.B. CA) überlassen. Sollte der Nutzer den Schlüssel selbst generieren, hat dies zum Vorteil, dass er die volle Kontrolle über den Erstellungsprozess besitzt. Jedoch obliegt ihm somit auch die Verantwortung den Prozess und die fertigen Schlüssel durch nötige Einstellungen im System und nötige Vorsicht zu sichern. Besonderes Augenmerk liegt dabei auf dem privaten Schlüssel, welcher unter keinen Umständen öffentlich werden darf. Da die Sicherheit der verschlüsselten Daten dann nicht mehr gewährleistet ist. Im Falle der Erzeugung durch eine zentrale Instanz gibt der Nutzer diese Verantwortung größtenteils aus der Hand. Dabei sollte er sich jedoch bewusst sein, dass so auch keine Kontrolle seinerseits besteht. Des Weiteren ist es nun notwendig den privaten Schlüssel sicher dem Nutzer

zur Verfügung zu stellen, was unter Umständen kritisch werden kann. Ist der Schlüssel erstmal erzeugt, muss dieser zwingend sicher aufbewahrt werden.

**Die Schlüsselspeicherung** muss gewährleisten, dass der private Schlüssel keinem Unbefugten zur Verfügung steht. Um ein Fehlverhalten des Nutzers auszuschließen, wäre es also ratsam nicht einmal diesem den Zugang zu dem privaten Schlüssel zu gestatten. Diesen Weg gehen Token (oder SmartCard). Token sind Chipkarten oder USBsticks bei denen der Schlüssel auf einem internen Chip sicher verwahrt wird. Der Zugriff ist ausschließlich über Sicherheitssoftware in Emailprogrammen (PGP) oder anderen (OpenSSH) möglich, welche den Schlüssel selbstständig auslesen und ihn nicht zwischenspeichern. Außerdem ist es natürlich möglich den Schlüssel selbst sicher zu verwahren.

**Die Verbreitung** des öffentlich Schlüssels ist über beliebige Wege möglich. Zum Einen besteht die Möglichkeit den öffentlichen Schlüssel auf einem KeyServer zu lagern oder ihn per E-Mail zu verschicken. Es gibt noch wesentlich mehr Alternativen zur Verbreitung, welche hier nicht alle genannt werden sollen. Allerdings ist die Authentizität des öffentlich Schlüssels zu berücksichtigen. Denn nicht immer lässt sich garantieren das der Schlüssel den man als öffentlichen Schlüssel von Person A gefunden oder erhalten hat auch wirklich von dieser stammt. Zu diesem Zweck werden Schlüssel von Gesprächspartnern, welche sich bereits gegenseitig überprüft haben, unterschrieben. Somit bestätigt der Signierende die Zugehörigkeit des öffentlichen Schlüssels zu Person A. Es wird eine mit dem privaten Schlüssel des Signierenden verschlüsselte Signatur an den öffentlichen Schlüssel von Person A angehängt. Aufgrund dessen unsere Operationen umkehrbar sind. Lässt sich nun diese Signatur mit dem öffentlichen Schlüssel des Unterzeichners öffnen und überprüfen. Da die Signatur jedoch nicht geändert werden kann, ist sie zudem vor Fälschung sicher. Dieses Verfahren wird ebenso bei Zertifikaten im WebOfTrust oder wie bereits erwähnt beim E-Mail Verkehr genutzt.

## 2.6 Hybride Kryptographie

Die hybride Kryptographie ist die Kombination von asymmetrischen und symmetrischen Verfahren. Eine symmetrische Verschlüsselung ist bei gleichem Sicherheitsgewinn im Gegensatz zu einer vergleichbaren asymmetrischen Verschlüsselung wesentlich effizienter zu berechnen. Allerdings muss je Kommunikationsbeginn ein erneuter aufwendiger Schlüsselaustausch (z.B. über Diffie-Hellmann) erfolgen, auch wenn mit dem Partner bereits in der Vergangenheit kommuniziert wurde. Die Lösung dafür stellt die hybride Kryptographie dar. Gehen wir davon aus es steht uns eine längere Konversation mit Person A bevor. Nun möchten wir nicht jede einzelne Nachricht mit einem asymmetrischen System verschlüsseln denn je nach Größe der Nachricht kann dies sehr Rechenintensiv werden. Um einen derartigen Aufwand zu umgehen legen wir einen symmetrischen Schlüssel fest, verschlüsseln diesen mit dem öffentli-

chen Schlüssel von Person A und übermitteln ihm diesen sicher verschlüsselt. Person A verfügt nun, nach der Entschlüsselung, ebenso wie wir über den symmetrischen Schlüssel über den nun die Nachrichten des Gespräches wesentlich effizienter verschlüsselt werden können. Nicht immer wird bei der hybriden Kryptographie ein symmetrischer Schlüssel von einem Gesprächspartner bestimmt. Es existieren viele unterschiedliche Implementierungen auf die hier aber nicht eingegangen werden soll.

## 3 Kryptographie mit Java

### 3.1 Java Cryptography Architecture

Die Java Cryptography Architecture ist ein Provider (Anbieter) für Kryptographische Systeme der Programmiersprache Java. Er dient zur Implementierung von:

1. digitalen Signaturen,
2. Hash's,
3. Zertifikaten (und deren Überprüfung),
4. Verschlüsselungen,
5. Schlüsselerzeugung und Management,
6. Zufallszahlen,
7. und vieles mehr [?]

Ausserdem beinhaltet das JCA noch andere Provider. In den folgenden Kapiteln wollen wir uns genauer mit den Punkten vier bis sechs befassen.

### 3.2 Zufallszahlen

Das JCA bietet zur Generierung von Zufallszahlen einen implementierten, nicht Quell-offenen Zufallszahlengenerator welcher in die Klasse *SecureRandom* eingebettet ist. Mittels des in Abbildung 2 gezeigten Aufrufes kann ein solches Zufallszahlen-Objekt generiert und später zur Erzeugung von Schlüsseln Verwendet werden.

Abbildung 2: Codebeispiel SecureRandom

```
SecureRandom random =  
    SecureRandom.getInstance("SHA1PRNG", "SUN");
```

Der Generator befindet sich im Provider *SUN* und trägt die Bezeichnung *SHA1PRNG*. Er funktioniert laut Angaben von Oracle wie folgt:



“This algorithm uses SHA-1 as the foundation of the PRNG. It computes the SHA-1 hash over a true-random seed value concatenated with a 64-bit counter which is incremented by 1 for each operation. From the 160-bit SHA-1 output, only 64 bits are used.” [?]

Da es sich hierbei um den einzigen Implementierten Zufallszahlengenerator im JCA handelt, ist es auch die Standard Auswahl wenn keine alternative bereitgestellt wurde.

### 3.3 Schlüsselgenerierung

Abbildung 3: Codebeispiel KeyPairGenerator

```
KeyPairGenerator keyGen =  
    KeyPairGenerator.getInstance( "RSA" );  
keyGen.initialize( int keysize );  
KeyPair keys = keyGen.generateKeyPair();
```

(KeyGenerator ist analog zu benutzen)

### 3.4 Schlüsselspeicherung

### 3.5 Schlüsseleinigung

### 3.6 Ver- und Entschlüsselung

## 4 Zusammenfassung

Zum Schluss bitte eine Zusammenfassung!

## Literatur

[Eck13] Claudia Eckert. *IT-Sicherheit : Konzepte - Verfahren - Protokolle*. Oldenburg, 2013.

Abbildung 4: Codebeispiel KeyAgreement

```
KeyAgreement aKeyAgree =
    KeyAgreement.getInstance("DH", "JCE");
KeyAgreement bKeyAgree =
    KeyAgreement.getInstance("DH", "JCE");

KeyPair aPair = keyGen.generateKeyPair();
KeyPair bPair = keyGen.generateKeyPair();

aKeyAgree.init(aPair.getPrivate());
bKeyAgree.init(bPair.getPrivate());

aKeyAgree.doPhase(bPair.getPublic(), true);
bKeyAgree.doPhase(aPair.getPublic(), true);

SecretKey aSecret =
    aKeyAgree.generateSecret("AES");
SecretKey bSecret =
    bKeyAgree.generateSecret("AES");
```

Abbildung 5: Codebeispiel KeyAgreement

```
Cipher cipher =
    Cipher.getInstance("RSA", "BC");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
cipher.update(message);
byte[] crypt = cipher.doFinal();
```