

# Übungsblatt 6

zur Vorlesung

## Verteilte Systeme/Ubiquitous Computing

Wintersemester 2007/2008

**Achtung:** Die Klausuranmeldung ist unter

<http://www.mobile.ifi.lmu.de/Vorlesungen/ss06/vs/>

freigeschaltet. Diese bleibt bis zum 03.02.2008, 23:59 Uhr (MESZ) geöffnet. Die Klausur findet am Donnerstag, dem 07.02.2008, in den Räumen der Theresienstraße 39 um 18:30 statt (Einlass ab 18:15). Für die Teilnahme ist eine Anmeldung zwingend erforderlich! Nähere Details zur Klausur finden sich auf der oben genannten Web-Site.

### Aufgabe 20: (H) Atomare Transaktionen in Verteilten Systemen

- Welche Eigenschaften besitzen Transaktionen?
- Welche Voraussetzungen müssen hierfür gegeben sein?
- Welche Transaktionsarten existieren in Verteilten Systemen?

**Antwort:**

- Transaktionen besitzen die folgenden vier Eigenschaften:
  - Atomar: Nach außen scheinen Transaktionen unteilbar.
  - Consistent: Transaktionen verletzen keine Systeminvarianten.
  - Isolated: Nebenläufige Transaktionen beeinflussen sich nicht gegenseitig.
  - Durable: Falls eine Transaktion ausgeführt wird, sind alle Änderungen dauerhaft.
- Folgende Voraussetzung für Transaktionen müssen gegeben sein:
  - Unsichere Kommunikation wird von der Software auf unteren Schichten behoben.
  - Stabiler Speicher durch z.B. redundante Datenspeicherung
  - Transaktionsprimitive müssen entweder vom Betriebssystem oder vom Laufzeitsystem einer Sprache zur Verfügung gestellt werden.
- Es existieren drei Arten von Transaktionen:
  - Flache Transaktion (flat transaction)
  - Geschachtelte Transaktionen (nested transactions) und

- Verteilte Transaktionen (distributed transactions)

Weitere Details zu Transaktionen finden sich im Kapitel 7.6 des Vorlesungsskripts.

## Aufgabe 21: (H) Grundlagen der Sicherheit in Verteilten Systemen

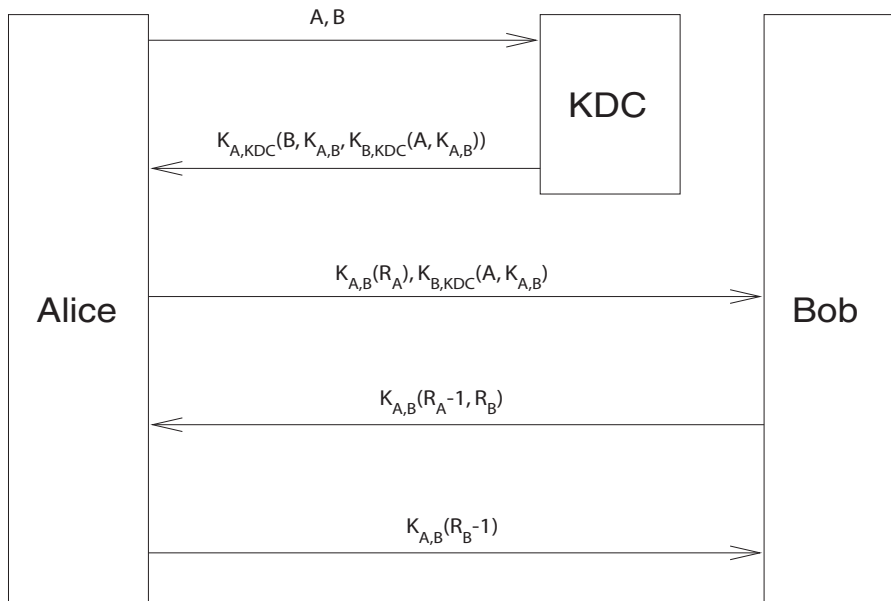
- Was ist der Unterschied zwischen Verfügbarkeit (Availability) und Zuverlässigkeit (Reliability)?
- Was ist der Unterschied zwischen Authentifizierung und Autorisierung?
- Warum müssen Authentifizierung und Nachrichtenintegrität immer gemeinsam gewährleistet werden, um sinnvoll zu sein?

### Antwort:

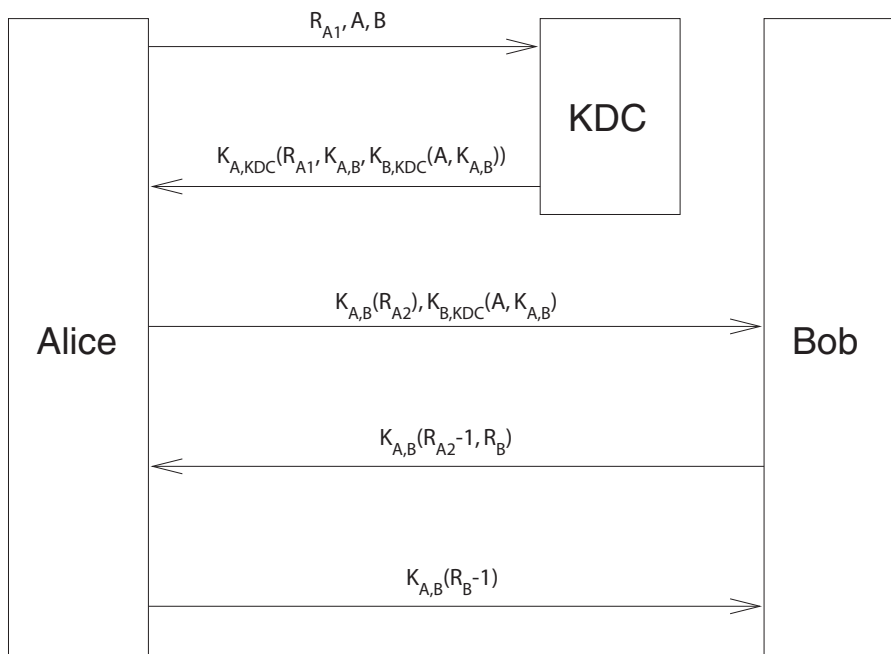
- Ein System, das 100%ig verfügbar ist, fällt nie aus und wird auch zu Wartungszwecken nie abgeschaltet. Ein 100%ig zuverlässiges System fällt nie aus, kann aber durchaus zeitweise un verfügbar sein, sofern die Unverfügbarkeit bewusst herbei geführt wurde, bspw. um Wartungsarbeiten durchzuführen.
- Die beiden Begriffe bezeichnen völlig unterschiedliche Zusammenhänge, auch wenn sie ähnlich klingen. Authentifizierung bewirkt, dass der Empfänger einer Nachricht sich sicher sein kann, dass diese wirklich vom behaupteten Sender stammt. Authentifizieren ist somit so etwas, wie sich auszuweisen. Autorisierung meint die Zuweisung von Rechten. Jemand wird autorisiert, etwas zu tun. Die beiden Begriffe hängen insofern zusammen, als dass jemand, der seine Rechte ausüben will, sich in der Regel zunächst authentifizieren muss.
- Es nützt nichts, wenn sich der Empfänger sicher sein kann, dass eine Nachricht wirklich vom behaupteten Absender stammt, wenn der Nachrichteninhalt verfälscht worden sein kann. Authentifikation ohne Nachrichtenintegrität könnte dazu führen, dass jemand eine Nachricht von der staatlichen Lotteriegesellschaft bekommt, dass er 1 Million Euro gewonnen hat. Er kann sich sicher sein, dass die Nachricht von der besagten Gesellschaft stammt. Allerdings schrieb ihm diese, dass er diesmal leider nichts gewonnen hat.

## Aufgabe 22: (H) Authentifizierung unter Verwendung eines Key Distribution Centers

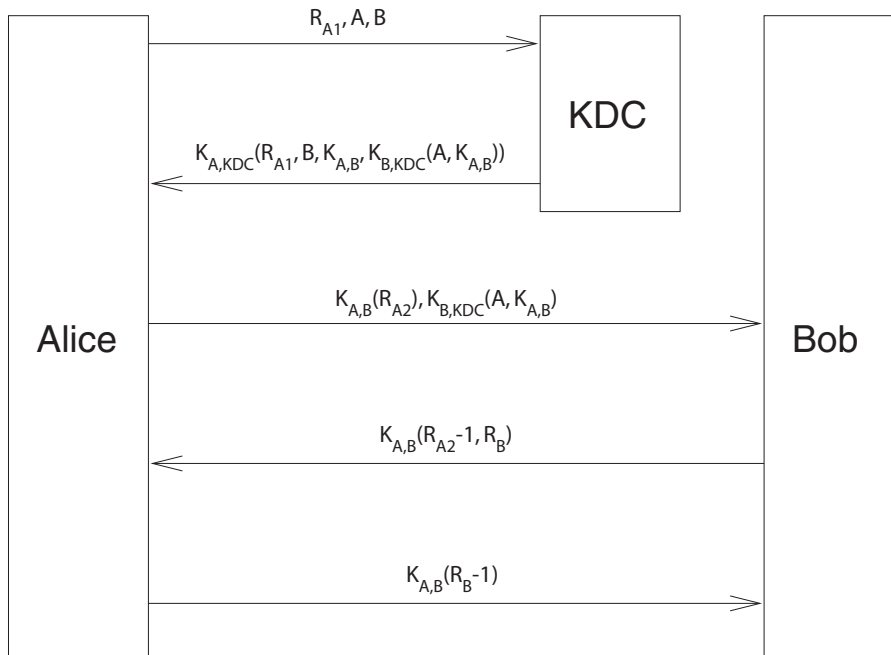
- a. Wie kann jemand, der auf irgendwelchen Wegen  $K_{B,KDC}$  herausbekommen hat, in den vermeintlich sicheren Kanal eindringen, der durch das folgende Protokoll hergestellt wurde?



- b. Wenn Bob weiß, dass einem Fremden sein Schlüssel  $K_{B,KDC}$  bekannt ist, kann er einen Angriff dadurch verhindern, dass er  $K_{B,KDC}$  ändert?
- c. Wie kann bei Verwendung des folgenden Protokolls selbst ohne Kenntnis von  $K_{B,KDC}$  ein Angriff erfolgen?

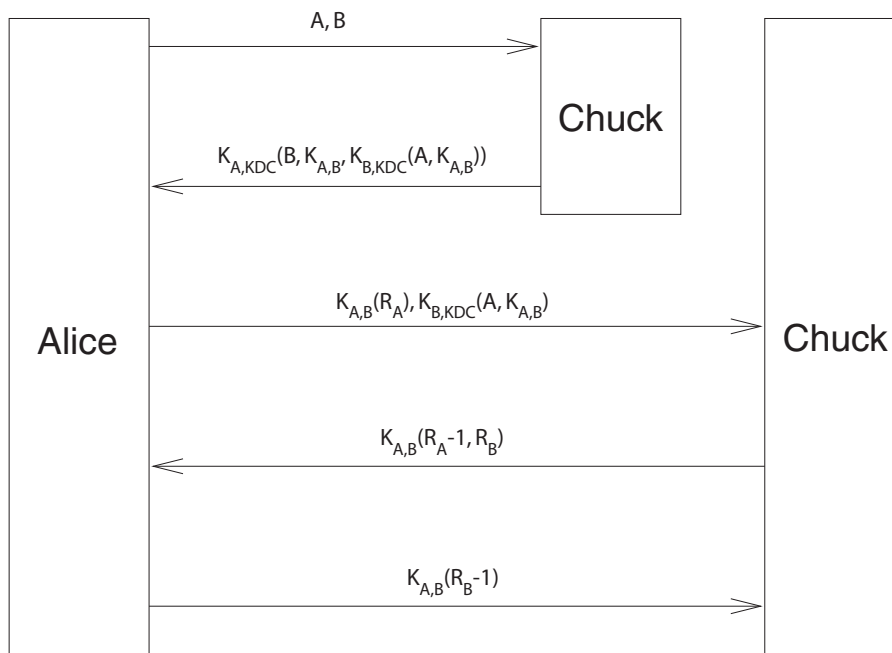


- d. Wo liegt die Schwachstelle des Needham-Schroeder-Protokolls, so wie es unten abgebildet ist? Wie kann diese Schwachstelle beseitigt werden?



**Antwort:**

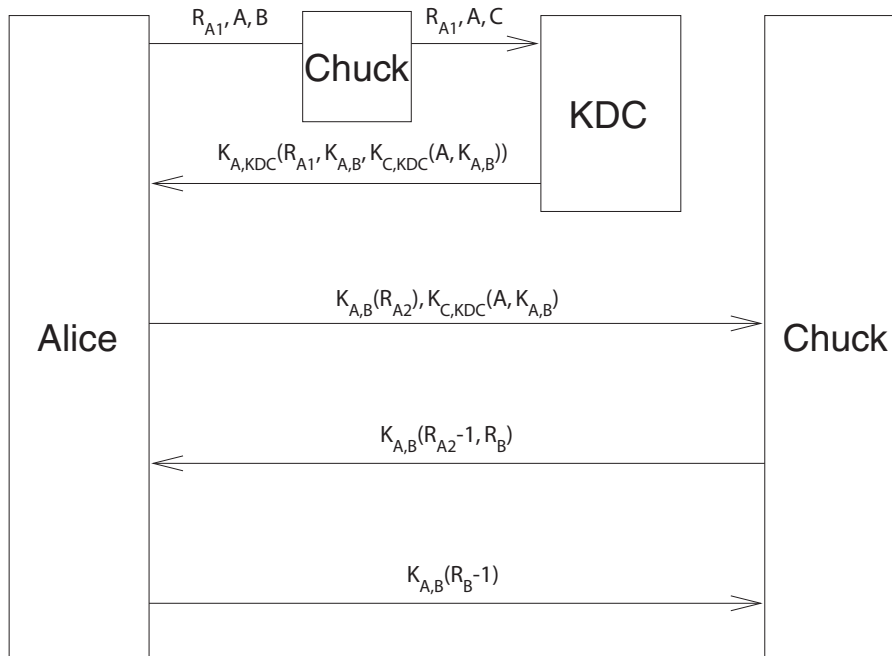
- a. Der Angreifer muss nur einmal die zweite Nachricht abhören. Danach wartet er geduldig darauf, dass Alice sich nochmal an Bob wenden möchte. Die Anfrage von Alice an den Key Distribution Center fängt der Angreifer ab und wiederholt die einst beobachtete Antwort des KDCs. Da er in Besitz von  $K_{B,KDC}$  ist, kann er aus der dritten Nachricht den Sitzungsschlüssel extrahieren und die Herausforderung  $R_A$  nach Alice' Zufriedenheit beantworten.



- b. Auch wenn Bob irgendwann seinen Schlüssel ändert, ist der Angriff trotzdem weiterhin möglich. Denn es kommt nur darauf an, dass der Angreifer in Besitz des Schlüssels  $K_{B,KDC}$  ist, der in der abgehörten

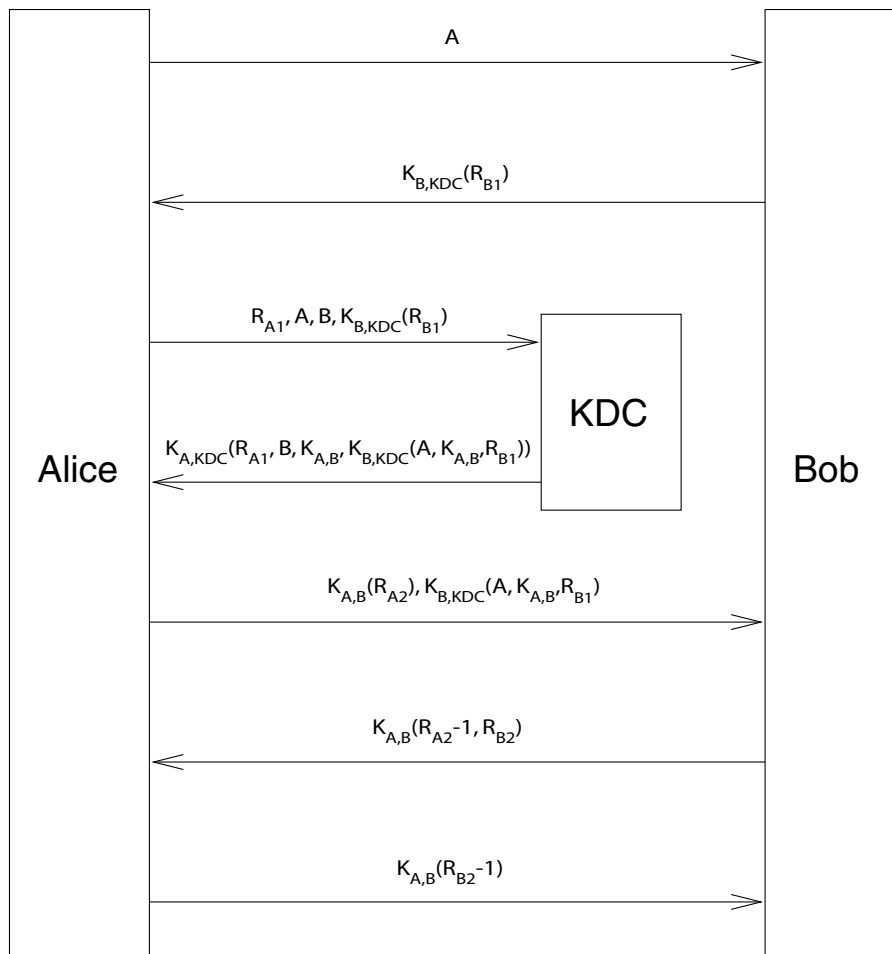
Antwort des KDCs verwendet wurde.

- c. Ein Angreifer kann in der Anfrage an den KDC einfach die Identität von Bob in seine eigene ändern.



- d. Die Schwachstelle des Needham-Schroeder-Protokolls liegt darin, dass ein Angreifer, der irgendwann mal in Besitz eines Sitzungsschlüssels  $K_{A,B}$  gekommen ist, einfach die dritte Nachricht wiederholen und so einen sicheren Kanal zu Bob aufbauen kann.

Diese Schwachstelle kann dadurch behoben werden, dass derjenige, der eine sichere Verbindung zu Bob aufbauen will, dort zunächst ein "Nonce" erfragt, der in das Ticket eingebaut wird. Dadurch kann Bob alte Tickets erkennen und den Aufbau einer sicheren Verbindung ablehnen.



### Aufgabe 23: (H) RSA

Der RSA-Algorithmus wurde nach seinen Erfindern Rivest, Shamir, Adleman benannt. Der öffentliche und der private Schlüssel werden nach diesem Algorithmus wie folgt bestimmt:

- Es werden zwei sehr große Primzahlen,  $p$  und  $q$ , bestimmt.
- Aus diesen wird  $n = p * q$  und  $z = (p - 1) * (q - 1)$  berechnet.
- Es wird eine Zahl  $e$  gewählt, so dass diese keinen gemeinsamen Teiler mit  $z$  besitzt.
- Darauf wird eine Zahl  $d$  ermittelt, die  $d * e \equiv 1 \pmod{z}$  erfüllt. Dies ist gleichbedeutend mit  $d = e^{-1} \pmod{z}$ .

Eins der beiden Zahlenpaare  $(d, n)$  und  $(e, n)$  kann als öffentlicher Schlüssel verwendet werden, während das andere als privater Schlüssel dient.

- Wählen Sie  $p = 47$  und  $q = 71$ . Berechnen Sie daraus  $n$  und  $z$ . Nehmen Sie für  $e$  die Zahl 79 an. Über den erweiterten Euklidischen Algorithmus kann daraus  $d$  mit 1019 bestimmt werden.
- Benutzen Sie  $e$  als privaten Schlüssel und kodieren Sie damit die Nachricht "6882326879666683". Dabei müssen Sie die Nachricht in Blöcke gleicher Länge zerteilen, die jeweils kleiner als  $n$  werden.

sind. Diese Blöcke transformieren Sie, indem Sie diese mit  $e$  exponentieren und Modulo  $n$  nehmen. Hängen Sie die Ergebnisse einfach aneinander, um die verschlüsselte Nachricht zu generieren. Für diese Rechnungen empfiehlt es sich, den “Basic Calculator” unter Unix zu verwenden, den Sie über den Befehl “bc” aufrufen können. (Modulo wird durch das Zeichen % repräsentiert.) Handelsübliche Taschenrechner sind durch die Größe der Ergebnisse überfordert und liefern falsche Ergebnisse.

- c. Entschlüsseln Sie die Nachricht, indem Sie die verschlüsselten Blöcke mit  $d$  potenzieren und anschließend eine Modulo- $n$ -Operation ausführen.

**Antwort:**

- a.  $p = 47$   
 $q = 71$   
 $n = 3337$   
 $z = 3220$   
 $e = 79$   
 $d = 1019$

- b. Zunächst muss die Nachricht in Blöcke gleicher Länge zerteilt werden, die kleiner als  $n$  sind. Es ergeben sich die folgenden Blöcke  $m_1$  bis  $m_6$ : “688”, “232”, “687”, “966”, “668”, “003”. Die Verschlüsselung erfolgt gemäß:

$$c_i = m_i^e \bmod n \quad (1)$$

Daraus ergibt sich:

$i$	$m_i$	$c_i$
1	688	1570
2	232	2756
3	687	2091
4	966	2276
5	668	2423
6	003	158

- c. Die verschlüsselten Blöcke können mit

$$m_i = c_i^d \bmod n \quad (2)$$

wieder entschlüsselt werden.

**Warum funktioniert RSA?**

Die Sicherheit des RSA-Algorithmuses beruht auf der Schwierigkeit, große Zahlen zu faktorisieren, gerade wenn die Faktoren wiederum groß sind. Deswegen wird  $n$  aus zwei großen Primzahlen gebildet.

Um zu verstehen, warum (2) wieder die ursprüngliche Nachricht erzeugt, müssen wir ein wenig in die Modulo-Algebra einsteigen. Hier gelten wieder das Kommutativ-, Assoziativ- und Distributivgesetz:

$$(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n \quad (3)$$

$$(a - b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n \quad (4)$$

$$(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n \quad (5)$$

$$a * (b + c) \bmod n = (((a * b) \bmod n) + ((a * c) \bmod n)) \bmod n \quad (6)$$

Wie bereits beschrieben, erfolgt die Entschlüsselung durch:  $m_i = c_i^d \bmod n$

Wenn man hier (1) einsetzt, ergibt sich:  $m_i = (m_i^e \bmod n)^d \bmod n$

Aufgrund von (5) kann dies umgeformt werden zu:

$$m_i = (m_i^e)^d \bmod n = m_i^{(e \cdot d)} \bmod n \quad (7)$$

d war so gewählt worden, dass gilt:

$$d \cdot e \equiv 1 \bmod z = 1 \bmod ((p-1) \cdot (q-1)) \quad (8)$$

Gleichzeitig gilt grundsätzlich:

$$x \equiv 1 \bmod y \equiv 1 + k \cdot y \quad (9)$$

für eine beliebige natürliche Zahl k.

(8) und (9) in (7) ergeben:

$$m_i = m_i^{(k \cdot (p-1) \cdot (q-1) + 1)} \bmod n = m_i \cdot m_i^{(k \cdot (p-1) \cdot (q-1))} \bmod n \quad (10)$$

(10) ist wahr, wenn

$$m_i^{(k \cdot (p-1) \cdot (q-1))} \bmod n = 1 \quad (11)$$

gilt.

(11) kann aufgrund von (5) umgeformt werden in:  $(m_i^{(p-1) \cdot (q-1)} \bmod n)^k = 1$ .

Dies ist gleichbedeutend mit:

$$m_i^{(p-1) \cdot (q-1)} \bmod (p \cdot q) = 1 \quad (12)$$

Der kleine Fermatsche Satz besagt, dass

$$a^{r-1} \bmod r = 1 \quad (13)$$

gilt, sofern r eine Primzahl und a kein Vielfaches von r ist.

Somit gilt:

$$m_i^{(p-1)} \bmod p = 1 \quad (14)$$

$$m_i^{(q-1)} \bmod q = 1 \quad (15)$$

Weiterhin gilt allgemein für Modulo-Operationen:

$$\text{Wenn } (u \equiv v \bmod p \wedge u \equiv v \bmod q), \text{ dann } (u \equiv v \bmod (p \cdot q)) \quad (16)$$

Die Kombination aus (14), (15) und (16) zeigt die Richtigkeit von (11) und damit (10). Somit wäre gezeigt, dass (2) die Nachricht entschlüsseln kann.

## Aufgabe 24: (T) IDEA

Der RSA-Algorithmus ist zwar sicherer als symmetrische Verfahren, benötigt aber 100 bis 200 mal mehr Rechenzeit als diese, sofern eine Implementierung in Software gewählt wurde. Bei Implementierungen in Hardware liegt dieser Faktor bei etwa 1000. Deswegen wird RSA in der Regel verwendet, um einen so genannten Sitzungsschlüssel auszutauschen, während die eigentliche Kommunikation dann mit Hilfe eines symmetrischen Verfahrens unter Verwendung dieses Sitzungsschlüssels erfolgt.

Das häufig verwendete Freeware-Produkt "Pretty Good Privacy" (PGP) von Philip Zimmermann benutzt beispielsweise das symmetrische IDEA-Verfahren (IDEA = International Data Encryption Algorithm) für die Kommunikation. Dieses soll im Rahmen eines Tutorials in der Übung exemplarisch vorgestellt werden.

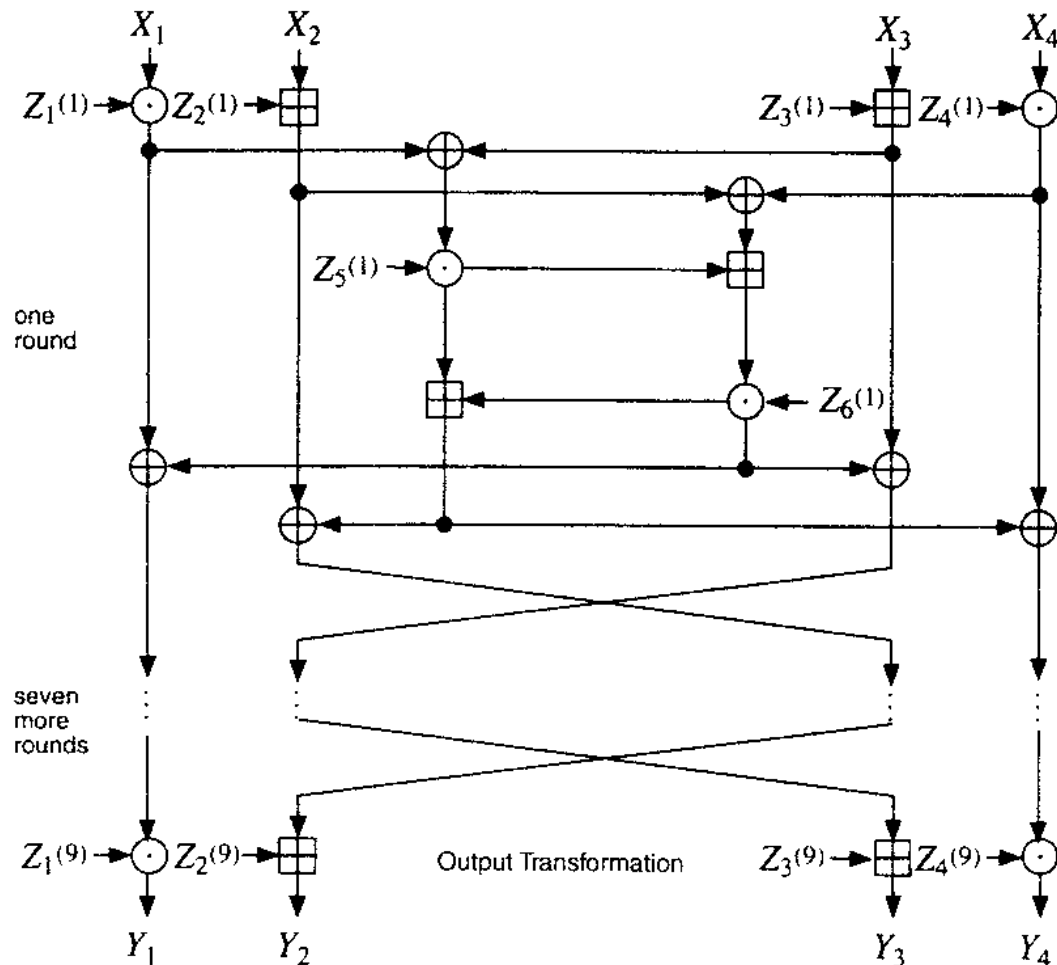
**Antwort:**

IDEA nimmt einen 64 bit langen Klartextblock und führt mit diesem unter Verwendung eines 128 bit langen Schlüssels drei Arten von Operationen aus:



- XOR
- Addition modulo  $2^{16}$
- Multiplikation modulo  $2^{16} + 1$

Die Reihenfolge der Operationen ist der folgenden Abbildung zu entnehmen:



$X_i$  bezeichnet den  $i$ -ten 16-bit-Block des Klartextes.

$Y_i$  symbolisiert den  $i$ -ten 16-bit-Block des verschlüsselten Textes.

$Z_i^{(r)}$  ist der  $i$ -te 16-bit-Block des Schlüssels in der  $r$ -ten Runde.

Der Kreis mit Kreuz steht für XOR,

das Quadrat mit Kreuz für eine Addition modulo  $2^{16}$ .

Der Kreis mit Punkt verkörpert die Multiplikation modulo  $2^{16} + 1$

Der Prozess einer Runde läuft wie folgt:

- $X_1$  und der erste Unterschlüssel werden multipliziert.
- $X_2$  und der zweite Unterschlüssel werden addiert.
- $X_3$  und der dritte Unterschlüssel werden addiert.
- $X_4$  und der vierte Unterschlüssel werden multipliziert.
- Die Ergebnisse von a. und c. werden einer XOR-Operation unterzogen.
- Ebenso findet eine XOR-Verknüpfung der Ergebnisse von b. und d. statt.

- g. Das Ergebnis von Schritt e. wird mit dem fünften Unterschlüssel multipliziert.
- h. f. und g. werden addiert.
- i. h. und der sechste Unterschlüssel werden multipliziert.
- j. g. und i. werden addiert.
- k. a. und i. werden XOR-verknüpft.
- l. c. und i. werden XOR-verknüpft.
- m. b. und j. werden XOR-verknüpft.
- n. d. und j. werden XOR-verknüpft.

Am Ende werden der zweite und der dritte Ergebnisblock vertauscht, bevor es in die nächste Runde geht. Insgesamt werden acht Runden durchlaufen. Nach der 8. Runde gibt es eine Endtransformation:

- a.  $X_1$  und der erste Unterschlüssel werden multipliziert.
- b.  $X_2$  und der zweite Unterschlüssel werden addiert.
- c.  $X_3$  und der dritte Unterschlüssel werden addiert.
- d.  $X_4$  und der vierte Unterschlüssel werden multipliziert.

Die Unterschlüssel werden so kreiert, dass der 128 bit lange Schlüssel in 8 Unterschlüssel geteilt wird. Nachdem diese 8 Unterschlüssel verbraucht wurden, wird der Originalschlüssel um 25 bit nach links rotiert, um dann wieder in 8 Teile für die nächsten 8 Unterschlüssel aufgeteilt zu werden. In einem ähnlichen Verfahren, das hier nicht näher betrachtet werden soll, werden aus demselben Schlüssel die Unterschlüssel für die Dekodierung gewonnen.