Inhaltsverzeichnis

1. RSA

RSA wird bei der asymmetrischen Verschlüsselung verwendet. Da die asymmetrische Verschlüsselung wesentlich langsamer als die symmetrische Verschlüsselung ist, wird sie in erster Linie zum Verschlüsseln von kurzen Texten verwendet. So kommt die asymmetrische Verschlüsselung beim **Schlüsselaustausch** für die symmetrische Verschlüsselung (s. später SSL/TSL) zum Einsatz.

1.1. Asymmetrisches Verschlüsselungsverfahren

Basis ist ein Schlüsselpaar.

1.1.1. Public- und Private Key

RSA ist ein asymmetrisches kryptographisches Verfahren, das ein **Schlüsselpaar** verwendet:

1. **public key (e … encrypt)** dient zum
    1. **Verschlüsseln** einer Nachricht
    2. **Überprüfen** der Unterschrift (signierten) einer Nachricht
2. **private key (d … decrypt)** dient zum:
    1. **Entschlüsseln** einer Nachricht m
    2. **Signieren** einer Nachricht m
3. **Schlüssellänge** (RSA-Modul n) (in Bits): **512, 768, 1024,2048, 4096 Bits**
   https://www.keylength.com/ enthält Empfehlungen über die diversen Schlüssellängen.

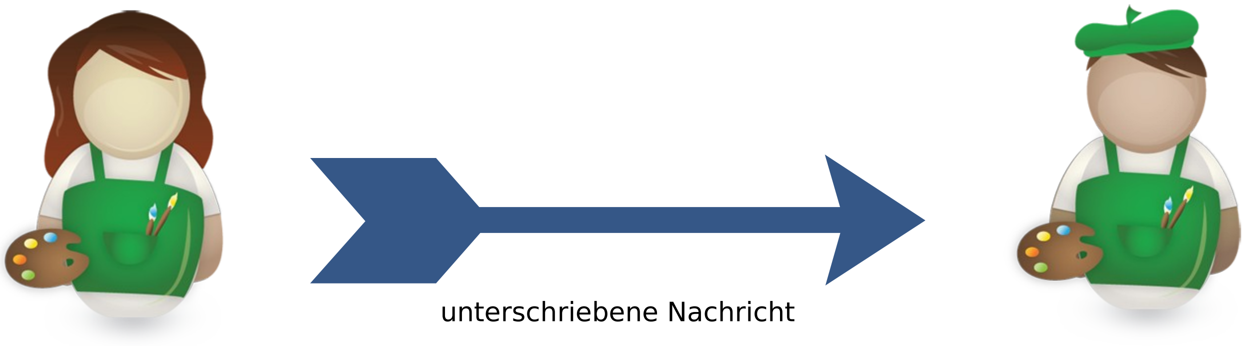
1.1.2. Verwendung

- Asym. Verschlüsselung ist ca. 800 mal langsamer als sym. Verschlüsselung (DES/AES), deshalb meist folg. Verwendung:
    1. **Signieren/Verfizieren (**Unterschreiben/Überprüfen der Unterschrift)
    2. **Verschlüsseln/Entschlüsseln**, ohne den Schlüssel austauschen zu müssen.
        - kurzer Nachrichten
        - Hashcodes(=Fingerprint) langer Nachrichten
    3. **Austausch des gemeinsamen Schlüssels** für die symm. Verschlüsselung (s.SSL).

1.1.3. Authentizität durch Unterschrift

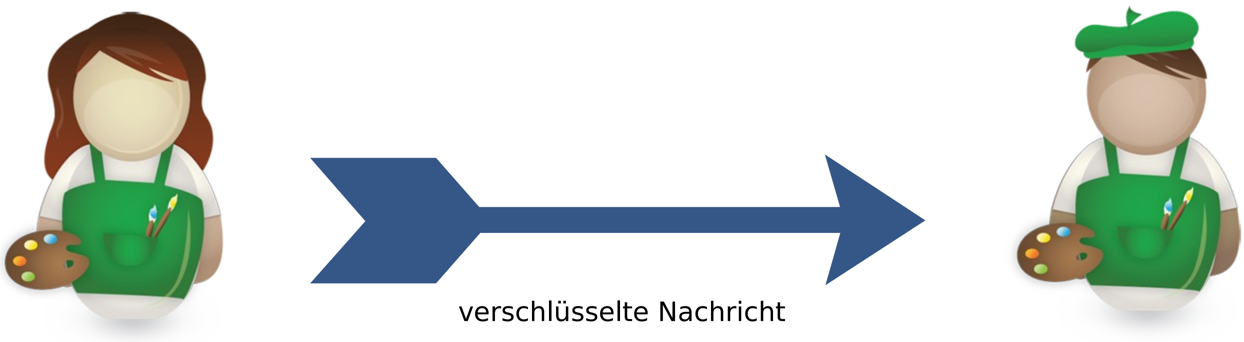
Authentizität durch Unterschrift

Alice unterschreibt/signiert Ihren Brief



1.1.4. Vertraulichkeit durch Verschlüsseln

Vertraulichkeit durch Verschlüsselung
Alice verschlüsselt Ihren Brief



Frage: Wer garantiert Alice, dass der Public-Key von Bob wirklich von Bob ist?

1.1.5. Schlüsselaustausch für die schnellere symmetrische Verschlüsselung

- Bei SSL wird die symmetrische Verschlüsselung verwendet (Performance).

1. Client: **fordert** den Public-Key des Servers an.
2. Client:
   1. **verschlüsselt** seinen symmetrischen Schlüssel mit dem Public-Key des Servers
   2. **sendet** diesen verschlüsselten symmetrischen Schlüssel an den Server
3. Server:
   Nur der Server kann diesen verschlüsselten symmetrischen Schlüssel mit seinem private-Key **entschlüsseln**.

Frage: Wer garantiert dem Client, dass der Public-Key des Servers wirklich vom Server stammt?

1.2. RSA: Grundidee

http://de.wikipedia.org/wiki/RSA-Kryptosystem

RSA : 1997, von Rivest, Shamir, Adleman

1.2.1. Einweg-Funktionen

☑ **RSA benutzt sogenannte Einweg-Funktionen**. (vgl. Einbahnstraßen)

Zum Verschlüsseln ist die Berechnung ganz **einfach.**
Zum Entschlüsseln (ohne Kenntnis der Schlüssel) **praktisch nicht** in ausreichender Zeit möglich. (Theoretisch aber durchaus möglich)

☑ **Eine Einweg-Funktion: Multiplikation von 2 Primzahlen**.

Verwenden Sie zB: http://pari.math.u-bordeaux.fr/gp.html

3259 * 5431
= 17699629

☑ **'Gegen die Einbahn:' Algorithmus: Primfaktorzerlegung (sehr langsam)!**

Gesucht sind alle Teiler der Zahl 17699629 ?

☑ **Übung**: Suche alle Teiler der Zahl 55141
factor(55141)

☑ Beispiele: (Mathematik-Software)
39-stellige Primzahl faktorisieren → < 1Sekunde
41-stellige Primzahl faktorisieren → > 8 Minuten
43-stellige Primzahl faktorisieren → > 19 Minuten
44-stellige Primzahl faktorisieren → Abbruch v. bekannter Mathematik-Software

Merke:

RSA basiert auf der Tatsache, dass man das

Produkt von zwei großen Primzahlen praktisch
nicht in vernünftiger Zeit wieder in die beiden
Primfaktoren

zerlegen kann.

1.2.2. RSA 768 Faktorisierung

aus: https://de.wikipedia.org/wiki/RSA_Factoring_Challenge

Die Faktorisierung dieser **232**-stelligen Zahl (RSA 768) wurde am 12. Dezember **2009** von Thorsten Kleinjung et al. vollendet.[1] Der RSA Factoring Challenge war zu dieser Zeit schon beendet, sodass kein Preisgeld ausgezahlt wurde.

RSA-768 =
1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

RSA-768 =
33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
*
36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

1.2.3. Falltür-Funktionen

Der Besitzer des privaten Schlüssels muss allerdings relativ einfach die Entschlüsselung durchführen können. Dazu verwendet man sogenannte Falltür-Funktionen, die **mit einer Zusatzinformation** sozusagen '**gegen die Einbahn' rechnen können**.
Rivest, Shamir und Adleman fanden einen mathematischen Zusammenhang.

Kurz gesagt:
Wenn man die Primfaktorzerlegung vom RSA-Modul kennt (sagen wir **p und q**) kann man **sehr einfach** den private-Key berechnen.

1.3. RSA: Begriffe

1.3.1. Schlüssel und Texte

Begriffe

m	☐ **Nachricht** (message) im Klartext ☐ Die Buchstaben der Nachricht müssen in Zahlen umgewandelt werden, sodass damit gerechnet werden kann. Beispiel: A → 01 , B → 02 , … oder A → 65 , B → 66 … ☐ Die zu verschlüsselnde Nachricht wird in Blöcke unterteilt. Diese Blöcke (in Bits) müssen kleiner als der RSA-Modul n (in Bits) sein. Ist dies nicht der Fall, muss man die Nachricht/Zahl in kleinere Blöcke zerlegen und diese dann einzeln verschlüsseln. Die Bitlänge des Blockes muss also kleiner sein als die Bitlänge des RSA-Moduls n.
c	cipher, der **Geheimtext**
p,q	2 sehr **große Primzahlen**
n	der öffentliche **RSA-Modul** (n= p*q)
e	der Verschlüsselungs-Exponent (encrypt)
d	der Entschlüsselungs-Exponent (decrypt)
(e,n)	**der Public-Key**
(d,n)	**der Private-Key**
c=m^e mod n	**Verschlüsseln**
m=c^d mod n	**Entschlüsseln**
s=m^d mod n	**Signieren**

m=s^e mod n	Verfizieren

1.3.2. Mathematisches I

☑ **Modulo / Restklassen:**

modulo liefert den Rest bei einer ganzzahligen Division:

20 mod 6 = 2 weil, 20 / 6= 3 mit 2 Rest Linearkombination: 20= 6*3 +2

43 mod 5 = 3

☑ **teilerfremd / ggT(a,b)=1**:

2 Zahlen sind teilerfremd, wenn ihr gemeinsamer größte Teiler 1 ist.

also:

ggT(23,15) = 1 → 23 und 15 sind teilerfremd

ggT(20,15) = 5 → 20 und 15 sind nicht teilerfremd

ggT(15,20) = 5 → 20 und 15 sind nicht teilerfremd

- ☐ Es gilt: **ggT(a,b) = ggT(b,a)**
  → Kommutativ-Gesetz
- ☐ Es gilt: **ggT(a,1) = 1**
- ☐ Es gilt: **ggT(a,b) = ggT(b, a mod b)**
- ☐ Es gilt: **ggT(a,b) = k*a + d*b** (mit k,d sind ganze Zahlen)
  → Der ggT läßt sich als **Linearkombination** darstellen.
  → k,d können mit dem '**Erweiterten Euklidischen Algorithmus**' berechnet werden(s.u.).
  Bsp: ggT(15,20) = 5
  5 = k*15 + d*20
  5 = 3*15 +(-2)*20
  5 = 45 -40
  5 = 5

☑ **Primzahlen**:

sind nur durch 1 und sich selbst teilbar und größer als 1.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,...

☑ **Phi-Funktion: (=Anzahl der zu n teilerfremden Zahlen)**

$$phi(n) := |\{a \in N | 1 \leq a \leq n \wedge ggT(a,n) = 1\}|$$

phi(n) ... Anzahl der zu n teilerfremden natürlichen Zahlen.
Betrachtet werden nur Zahlen von 1 bis n.

☑ Beispiele:
- ☐ phi(6) = 2 ,
  weil nur 1 und 5 zu 6 teilerfremd sind.

{1,2,3,4,5,6}

- wenn n eine Primzahl ist, dann gilt: **phi(n)= n-1**
  phi(13)=12,
  weil alle Zahlen von 1 bis 12 zu 13 teilerfremd sind.

- Wenn n das Produkt von 2 Primzahlen ist, dann gilt: **phi(p*q)= phi(p) * phi(q)**
  phi(10)= phi(2*5)= phi(2)*phi(5)= (2-1) * (5-1)= 4, weil
  phi(10)=
  streiche alle Zahlen, die einen gemeinsamen Teiler (>1) mit 10 haben:
  {1,~~2~~,3,~~4~~,~~5~~,~~6~~,7,~~8~~,9,~~10~~}. Es bleiben: {1,3,7,9}. Also 4 Stück.

- **Allgemein:**
  phi(p*q)= (p-1)*(q-1) , wenn p,q prim

- **Merke:**
  phi(p*q) wird zur Berechnung von e und d
  phi(p*q)= (p-1)*(q-1) verwendet,
  also das Produkt von 2 sehr großen Primfaktoren.

- **Die Berechnung von e und d ist also trivial, wenn p und q bekannt sind, sonst praktisch unmöglich.**

  (Beweis und Berechnung von e,d siehe unten.)

1.4. RSA: Beispiel mit kleinen Zahlen

"Bob soll an Alice eine verschlüsselte Nachricht schicken."

Schritte:

1. Alice erzeugt ein Schlüsselpaar (e,n) und (d,n)
2. Bob verwendet den public-key (e,n) von Alice und
   verschlüsselt die Nachricht m und erzeugt den Cipher-Text c mit: c= m^e mod n
3. Alice entschlüsselt c mit: m= c^d mod n

1.4.1. Alice erzeugt ein Schlüsselpaar (e,n) und (d,n)

1. **Wähle zwei große unterschiedliche Primzahlen.**	**p,q … prim**	**p=13** **q= 7**
2. **Berechne den RSA Modul n** zum Ver/Entschlüsseln	**n=p*q**	**n= 13*7=91**
3. **Berechne phi(n)** zum Ermitteln der Schlüssel-Exponenten e,d	**phi(n)=** **phi(p*q)=(p-1)*(q-1)**	**phi(n)=** **phi(13*7)=12*6=72**
4. **Wähle den**	**1< e < phi(n)**	**e= 5**

Verschlüsselungs-Exponenten e:	und **ggt(e, phi(n)) = 1**	

5. **Berechne den Private-Key (=Entschlüsselungs-Exponenten) d**	**e*d≡ 1 mod ((p−1)(q−1))**
	
	oder
	e*d ≡ 1 mod (phi (p*q))
	
	oder
	e*d ≡ 1 mod (phi (n))
		5 * d ≡ 1 mod (12*6)
		
		5 * d ≡ 1 mod (72)
		
		Versuch 1:
		d durch Einsetzen berechnen:
		5*1 mod 72=
		5*2 mod 72=
		…
		29*2 mod 72= 1
		Hier ein python-script:
		
		`$> python python-rsa-calc-d.py`
		
		`e=5`
		`phi=72`
		`for d in range(100):`
		`    if (d*e)%phi == 1:`
		`        print d`
		
		→ d= 29
		
		Versuch 2: (besser)
		d durch den Erweiterten Euklidischen Algo. (s. unten) berechnen:
		http://pari.math.u-bordeaux.fr/gp.html
		
		gcdext(5,72)
		→ 29
		
		5*29 mod 72= 1
		→ d= 29

Ergebnis		
1. Public-Key	**(e,n)**	**(5,91)**
2. Private-Key	**(d,n)**	**(29,91)**

Merke:

- Zum eigentlichen **Ver/Entschlüsseln** wird der **RSA-Modul n** (=p*q) verwendet.
- Zum Berechnen der **Exponenten e und d** wird **phi(n)**= phi(p*q)=(p-1).(q-1) verwendet.
- Zum Berechnen von d kann man den Erweiterten Euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teilers verwenden. https://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus (s. unten)

Hinweise:

- e: Aus Effizienzgründen wird e klein gewählt, üblich ist die 4. Fermat-Zahl 2^{16}= 65535
- Folgende Zahlen wurden also berechnet: (Durchgestrichene dürfen auf keinen Fall öffentlich gemacht werden!!!)
- ~~**p.............13**~~
  ~~**q...............7**~~
  ~~**phi(n).....72**~~
  ~~**d.............29**~~

- **n............91**
  e..............5

1.4.2. Bob verschlüsselt

Die Nachricht muss 'aufbereitet' werden. In unserem Beispiel ordnen wir den Buchstaben Zahlen zu. So könnte man die Ascii-Werte der Zeichen verwenden.

Wir wollen hier allerdings folgende Zuordnung wählen:
A→1, B→2, C→3, D→4, E→5, F→6, G→7, H→8, I→9, J→10,K→11, L→12, M→13, …

Bob will die Nachricht m="GEHEIM" verschlüsselt an Alice senden.

- Der Public-Key von Alice lautet: **(e,n)= (5,91)**
- Verschlüsselt wird mit: **c= m^e (mod n)**
- Zum Rechnen verwende**: http://pari.math.u-bordeaux.fr/gp.html**

Nachricht m	G	E	H	E	I	M
aufbereitete Nachricht: m	7	5	8	5	9	13
c= m^e (mod 91)	7^5%91	5^5%91	8^5%91	5^5%91	9^5%91	13^5%91
cipher	63	31	8	31	81	13

1.4.3. Alice entschlüsselt

- Der Private-Key von Alice lautet: **(d,n)= (29,91)**
- Entschlüsselt wird mit: **m= c^d (mod n)**

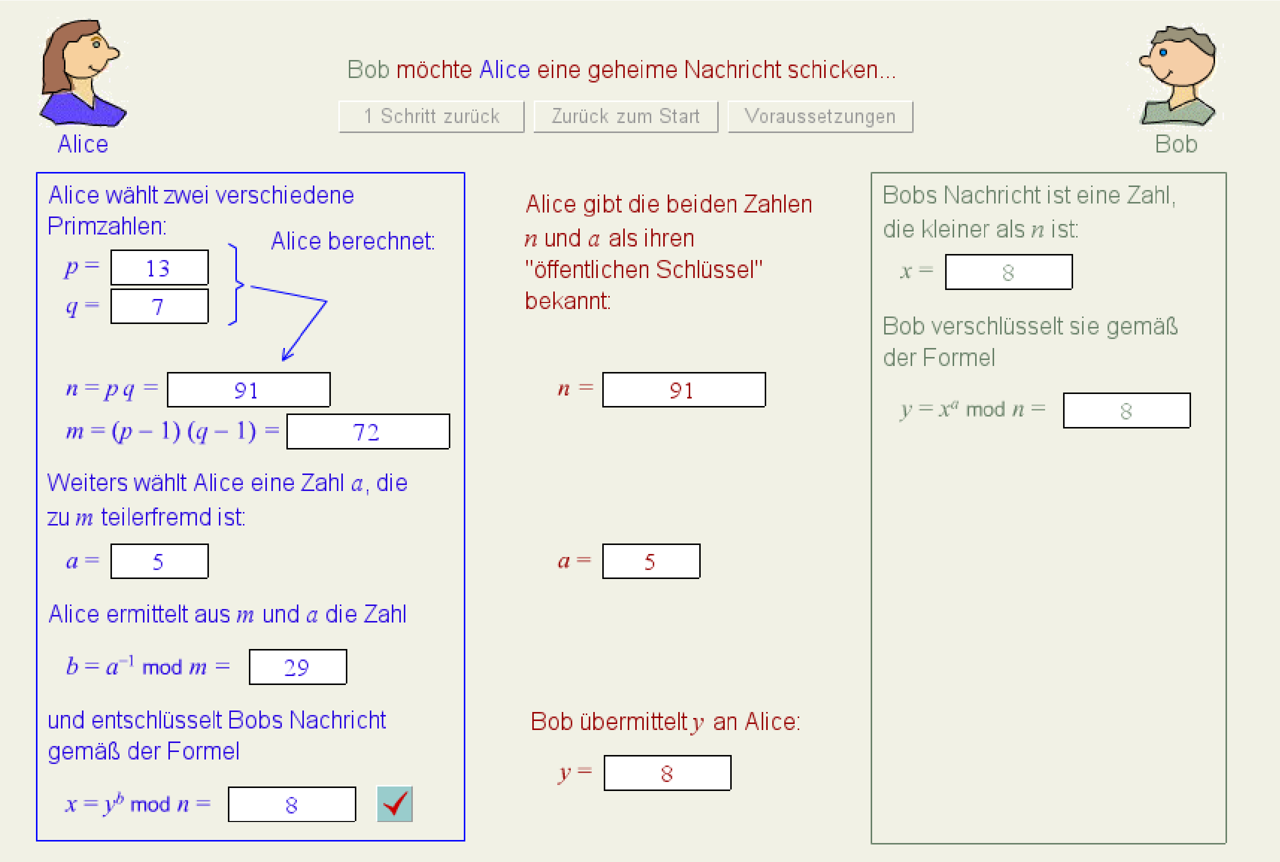
cipher: c	63	31	8	31	81	13
m=c^d mod 91	63^29%91	31^29%91	8^29%91	31^29%91	81^29%91	13^29%91
m= c^d mod 91	7	5	8	5	9	13
Nachricht m:	G	E	H	E	I	M

1.4.4. Übung: Web Animation

Die folgende Animation zeigt sehr anschaulich nochmals das RSA-Verfahren.

http://www.mathe-online.at/materialien/Franz.Embacher/files/RSA/

p=13, q= 7, n= 91, phi(n)= 72, e=5, d=29,



1.5. Den Entschlüsselungs-Exponenten d berechnen

Aufgabenstellung:

Bekannt sind

1. **e= 5**
2. **phi(n)= 72**
3. Formel zur Berechnung von d: **e*d ≡ 1 mod (phi (n))**
   5*d ≡ 1 mod (72)
   Das bedeutet, e und d heben sich auf (ver-/entschlüsseln), genauso wie 4 und 1/4 bezüglich Multiplikation (4 · 1/4 = 1).
   Man sagt auch: d ist die zu e modular inverse Zahl.
4. **5*d ≡ 1 mod (72)** hat
   -entsprechend der **Lösbarkeit von linearen Kongruenzen** (s.unten) -
   eine Lösung, wenn **ggT(e,phi(n)) = 1** ist.
   ggT(5,72) = 1

   Dies ist der Fall, weil bei der Wahl von e die Bedingungen gelten:
   1<e<phi(n)   und **ggT(e,phi(n))=1**
5. Um d zu berechnen, verwende nun die **Linearkombination von ggT(e,phi(n))=1**:

   1= ggT(e,phi(n))= **k***phi(n) + **d***e

   1= ggT(5,72)= k*72 + d*5
6. k und d werden nun durch den **Erweiterten Euklidischen Algorithmus** berechnet.

1.5.1. Der erweiterte euklidische Algorithmus

https://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus

1. Berechne d und k für die Gleichung:
   `1= ggT(5,72) = k*72 + d*5`
2. Vorgehen:
   Berechne den ggT(5,72) und merke -pro Rechenvorgang- die Linearkombination.
   Wenn der ggT() den Wert 1 liefert, verwende die Methode des 'Rückwärts Einsetzens'.
3. es gilt u.a.:
   1. ggT(e,phi(n)) = ggT(phi(n),e)
      ggT(5,72) = ggT(72,5)
   2. ggT(a,b)= ggT(b, a%b)
   3. ggT(a,1)= 1

ggT berechnen	Division	modulo	Linearkombination	Rest explizit
ggT(72,5)	72/5=14	72%5=2	72=14*5+2	2= 72 - 14*5
ggT(5,2)	5/2=2	5%2=1	5=2*2+1	1= 5 - 2*2
ggT (2,1)=1				
				rückwärts einsetzen
				1= 5-2*2
				1= 5 – 2(72 -14*5)
				1= (-2)*72 +(29)*5
				k=-2 d= 29

→ Die Lösung: **d= 29**

Probe:

5*d ≡ 1 mod (72)

5*29 ≡ 1 mod (72)

145%72= 1

w.z.b.w

1.5.2. Zusammenfassung: d berechnen

Gegeben: e=5, phi(n)=72
Gesucht: d
Lösung: 1= ggT(5,72)= k*72 + 5*d
Lösung allgemein: 1= ggT(e,phi(n))= k*phi(n) + d.e

1.6. RSA: Fragen

Für RSA sind folgende Angaben bekannt: **p= 7, q= 11, e= 17**.

1.6.1. Was gilt für e=17, sodass ein privater Schlüssel d existiert?

Allgemein:
Für e muss gelten:

1. 1 < e < phi(n)
2. ggt(e,phi(n))=1 (teilerfremd)

Im Speziellen:
Berechne phi(n)= phi(p*q)= (p-1)*(q-1)= 6*10= 60
☑ 1 < 17 < 60
☑ ggt(17,60)=1

Antwort:
e=17 erfüllt somit alle Anforderungen.

1.6.2. Bestimmen Sie den privaten Schlüssel d.

Allgemein:
Es gilt: e*d ≡ 1 mod phi(n)

Im Speziellen:
17*d ≡ 1 mod 60 ,

Antwort:
Entspechend der linearen Kongruenz ist eine Gleichung der obigen Form nur dann für d lösbar, wenn der ggT(17,60)=1.

Weiters liefert der erweiterte Euklidische Algorithmus zum ggT(17,60) eine Linearkombination der folgenden Art:
ggT(17,60) = k*60 + d*17

Es gilt also:
17*d ≡ 1 mod 60 hat genau dann eine Lösung, wenn:

ggT(e, phi(n)) = ggT(17,60)=1 , d.h. k*60 + d*17= 1

Berechnung von d mit dem erweiterten eukl. Algorithmus:

1. ggT(e,phi(n)) = ggT(phi(n),e)
   ggT(17,60) = ggT(60,17)
2. ggT(60,17) = ggT(17, 60%17)
   ...

ggT berechnen	division	modulo	Linearkombination	Rest
ggT(60,17)	60/17=3	60%17=9	60=3*17+9	9= 60 - 3*17
ggT(17,9)	17/9=1	17%9=8	17=1*9+8	8= 17 - 1*9
ggT(9,8)	9/8=1	9%8=1	9=1*8+1	1= 9 - 1*8
ggT(9,1,)=1				
				rückwärts einsetzen

				1= 9-8
				1= 9 – (17 -9)
				1= 2*9 - 17
				1= 2*(60-3*17) -17
				1= 2*(60) -6*17 -17
				1= 2*(60) -7*(17)
				2*(60) -7*(17)
				k=2 d= -7

Um d positiv zu erhalten, kann man -wegen der Restklasse mod 60 – zu d 1*60 oder 2*60 … addieren. Also zum Beispiel -7+60= 53.

Demnach gilt:
17*-7 ≡ 1 mod 60
aber auch
17*53 ≡ 1 mod 60
usw.
Somit ist d= 53 eine Lösung.
Probe: (17*53) mod 60 = 1

1.6.3. Welcher Zahlenbereich kann als Nachricht verwendet werden?

Antwort:
Die Nachricht m muss im Bereich: 0 < m < n.

Da n=p*q= 7*11= 77 muss die Nachricht m<77 sein.

1.6.4. Wie lautet die Chiffrezahl c für die Nachricht m=66?

Es gilt: c= m^e mod n
c= 66^{17} mod 77
c= 33

1.6.5. Zeigen Sie, dass die Entschlüsselung von c wieder zu m führt.

Es gilt: m= c^d mod n
m= 33^{53} mod 77
m= 66

1.6.6. Welche der folgenden Zahlen müssen geheim gehalten werden?

Annahme: Folgende Zahlen wurden berechnet: Streichen Sie die Zahlen durch, die auf keinen Fall öffentlich gemacht werden dürfen.

p............13
q..............7
n............91
ph.i(n)....72
e..............5
d............29

1.6.7. Weitere Fragen

- ☐ n,p,q sind Zahlen, die geheim, öffentlich, prim oder nicht prim sein können? Was gilt wirklich.

  n: O prim, O nicht prim, O geheim, O öffentlich
  p: O prim, O nicht prim, O geheim, O öffentlich
  q: O prim, O nicht prim, O geheim, O öffentlich

- ☐ Wie können p und q geheim sein, wenn doch n= pq öffentlich bekannt ist?
  Antwort:
  Dies beruht nur darauf, dass die Primfaktorzerlegung von n zu rechenaufwändig ist, da n sehr groß ist (z.B. 4096 Bit lang).

- ☐ Für die Zahl **e** , den öffentlichen Schlüssel, muss gelten
  ggt(e, phi(n))= _____________ Antwort: 1

  Hierbei ist phi(n)=___________ Antwort: (p-1)*(q-1)
  die Anzahl der zu **n** teilerfremden Zahlen, die kleiner als **n** sind.

- ☐ Gib eine math. Erklärung für **phi(n) = (p − 1)(q − 1)**
  Antwort: n ist das Produkt aus 2 großen Primzahlen (p und q) und wird wie folgt berechnet: Es gilt:

  phi(p*q)=phi(p)*phi(q)= (p-1)*(q-1)

- ☐ Wie wählt man d und e?
  e wird gewählt: 1<e<phi(n) und ggt(e,phi(n))=1
  d wird mit dem erweiterter eukl. Algorithmus berechnet.
  Es gilt: e*d ≡ 1 mod (phi(n))

1.7. RSA: Zusammenfassung

1.7.1. RSA: Im Überblick

1. **Wähle zwei große Primzahlen mit p!=q:**
   p , q
2. **Berechne den RSA Modul n:**
   n = p * q
3. **Berechne die Eulersche Phi-Funktion:**
   phi(n)= phi(p*q) = **(p−1)(q−1)**
4. **Wähle Verschlüsselungsexponenten e:**
   1< e <phi(n) und **ggT(e,phi(n))= 1**
5. **Berechne Entschlüsselungsexponenten d:**
   e * **d ≡ 1 mod (phi(n))**
   1= ggT(e,phi(n))= k*phi(n) + d*e

Hinweise:
- p,q, phi(n): werden nicht mehr benötigt.

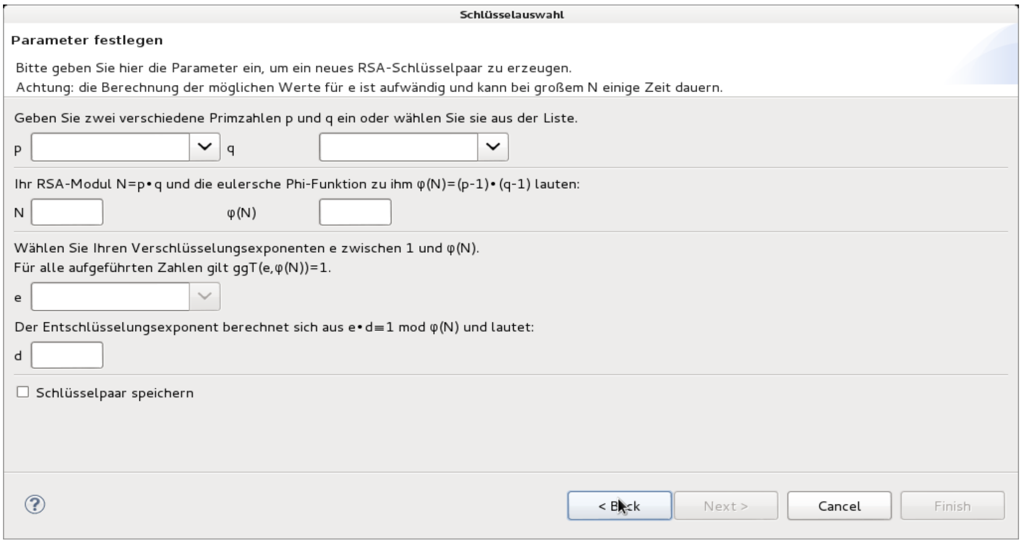
- e: Aus Effizienzgründen wird e klein gewählt, üblich ist die 4. Fermat-Zahl 2^{16}= 65535
- d: Zur Berechnung wird der Erweiterte Euklidische Algorithmus verwendet. Es gilt:
  - ggT(e,phi(n)) = ggT(phi(n),e)
  - ggT(a,1)=1
  - ggT(a,b)= ggT(b, a mod b)

1.7.2. +Übung: JCrypTool installieren

http://www.cryptool.org/

p=13, q= 7, n= 91, phi(n)= 72, e=5, d=29,

Probieren Sie das obige Beispiel mit dem Programm: JCrypTool
Installieren Sie JCrypTool
Menu:Visualisierungen:RSA-Kryptosystem
→ Schlüsselwahl
→ neues Schlüsselpaar
...



1.8. +Mathematik II

1.8.1. Lösbarkeit von Linearen Kongruenz

https://de.wikipedia.org/wiki/Kongruenz_(Zahlentheorie)

1. **Lösung (allgemein):** Lösbarkeit von linearen Kongruenzen:

   Eine lineare Kongruenz der Form

   `a*x ≡ c (mod m)` ist genau dann in **x** lösbar, wenn
   `g= ggT(a,m)` die Zahl **c teilt**.

   In diesem Fall besitzt die Konruenz **genau g Lösungen** in {0,1,2, ..., m-1}.

   Der erweiterte euklidische Algorithmus - angewendet auf a und m - kann zur Berechnung der Lösungen herangezogen werden.

   `g= ggT(a,m) = d*a + k*m`

Denn dieser Algorithmus berechnet auch d und k.

Somit lautet eine Lösung dann: **x1= (d*c)/g**

Beispiel 1: **4*x ≡ 10 mod (18)** ist lösbar, weil ggT(4,18)= 2 teilt die Zahl 10. Es gibt 2 Lösungen. Die Lösungen können durch den erweitertem euklidische Algorithmus gefunden werden.(s.u.)

2. **Lösung (Beispiel):** Lösbarkeit von linearen Kongruenzen:

   5*x ≡ 1 mod (72) ist lösbar, weil ggT(5,72)= 1 teilt die Zahl 1.

   Es gibt also genau 1 Lösung. Die Lösung kann durch den erweitertem euklidische Algorithmus gefunden werden.

   1= ggT(5,72) = d*5 + k*72

   Denn dieser Algorithmus berechnet auch d und k.

   Somit lautet die Lösung dann: x= (d*c)/g. Also x= (d*1)/1.

   → Die Lösung x=d. d wird durch den erweiterten euklidischen Algorithmus berechnet.

1.8.2. Mit große Zahlen rechnen

☑ Die Methode des fortgesetzten Quadrierens:

- ☐ Beim Potenzieren entstehen beim RSA Algorithmus gigantische große Zahlen. Bereits bei kleinen Beispielen entstehen zB. mit 1697^{157} eine Zahl mit über 500 Ziffern.

  Bei den im praktischen Einsatz befindlichen Schlüssellängen und Blockgrößen entstehen Werte, die ausgeschrieben mehrere Seiten füllen.

- ☐ Um das Problem der zu großen Zahlen in den Griff zu bekommen, verwenden Programmierer das Verfahren des fortgesetzten Quadrierens.

☑ Dieses Verfahren macht zum Einen davon Gebrauch, dass mehrfaches Quadrieren sehr schnell sehr hohe Potenzen erzeugt, so ist zum Beispiel

$$17^{67} = 17^{64} * 17*17*17 = ((((((17^2)^2)^2)^2)^2)^2) * 17 * 17 * 17$$

☑ **siehe auch: https://de.wikipedia.org/wiki/Bin%C3%A4re_Exponentiation**

- ☐ Zum Anderen möchte man ja gar nicht den Wert 17^{67} wissen, sondern nur den modularen Rest 17^{67} (mod n).

  Dabei ändert sich der Rest nicht, wenn bei jedem Schritt des Quadrierens modulo gerechnet wird:

  $$17^{67} \text{ (mod n)}= ((((((17^2 \text{(mod n)})^2 \text{(mod n)})^2 \text{ (mod n)})^2 \text{ (mod n)})^2 \text{ (mod n)})^2 \text{ (mod n)}) * 17 * 17 * 17 \text{ (mod n)}$$

  Auf diese Weise muss man nur mit Zwischenwerten arbeiten, die in etwa so groß sind wie n.

- ☐ weiters gilt, wenn Φ(n) und e teilerfremd sind:

$$p^e \equiv p^{e \bmod \phi(n)} (\bmod n)$$

d.h. der Aufwand für das Potenzieren kann verringert werden.

1.9. +RSA in Java

http://www.codeplanet.eu/tutorials/java/7-aes-und-rsa-in-java.html

1.10. Weitere Quellen

☑ Quelle:
- ☐ http://www.cryptool.org/ (super)
- ☐ http://hayageek.com/rsa-encryption-decryption-openssl-c/ (super)
- ☐ http://www.iti.fh-flensburg.de/lang/krypto/index.htm
- ☐ http://verplant.org/facharbeit/html/node3.html
- ☐ http://ddi.cs.uni-potsdam.de/HyFISCH/Informieren/Theorie/KryptoHess/Krypto1.html
- ☐ https://mathematik.de/ger/information/wasistmathematik/rsa/rsa.html
- ☐ http://www.easy-coding.de/wiki/java/ver-und-entschluesselung-mittels-rsa-u-aes-in-java.html
- ☐ http://www.example-code.com/vcpp/rsa.asp
- ☐ http://www.di-mgt.com.au/rsa_alg.html (super)
- ☐ http://www.di-mgt.com.au/bigdigits.html (C BigInteger)
- ☐ http://www.codeplanet.eu/tutorials/java/7-aes-und-rsa-in-java.html
- ☐ http://www.arndt-bruenner.de/mathe/scripts/chinesischerRestsatz.htm
- ☐ http://www.sagemath.org/doc/thematic_tutorials/numtheory_rsa.html