TECHNISCHE UNIVERSITÄT DRESDEN

ProSeminar Kryptografie

Prof. Dr. Ulrike Baumann

WS 2006/2007

RSA-Verschlüsselung

Francesco Kriegel

14. 12. 2006

Inhaltsverzeichnis

1 Public-Key-Verfahren

1.1 Idee

Die Idee der Public-Key-Kryptosysteme kommt von

Whitfield Diffie und *Martin Hellman,*

beide veröffentlichten im Jahre 1976 ihre Arbeit

„New Directions in Cryptography“.

Dazu stelle ich folgendes Zitat vor.

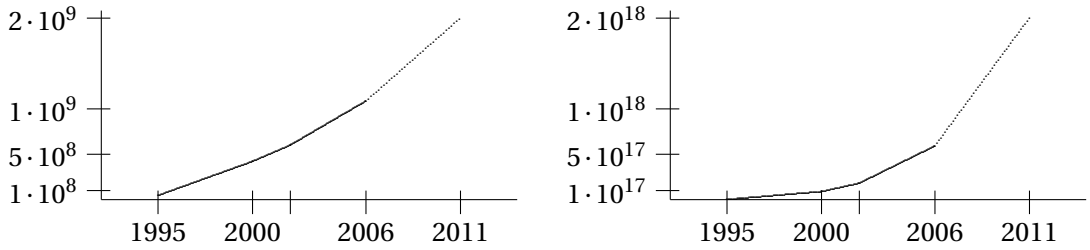
> Die Public-Key-Kryptografie wurde im Mai 1976 als Kind von zwei Problemen geboren, dem Schlüsselverteilungsproblem und dem Signaturproblem. Die Entdeckung bestand nicht in einer Lösung, sondern in der Erkenntnis, dass die beiden Probleme, die beide aufgrund ihrer Definition unlösbar schienen – überhaupt lösbar sind und dass beide Probleme gemeinsam gelöst werden. (W. Diffie)

Diffie nennt hier zwei Probleme, die bei den Private-Key-Verfahren auftreten:

(1) Schlüsselverteilungsproblem

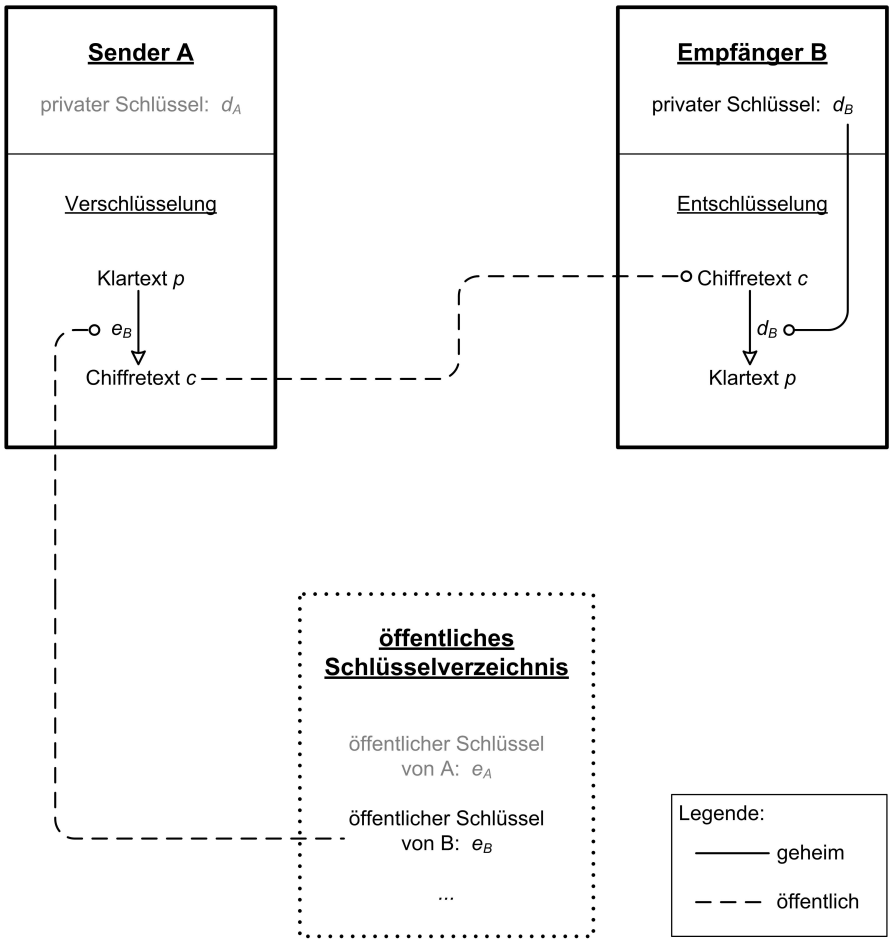
(2) Signaturproblem

Beide Missstände beruhen auf der Symmetrie der Private-Key-Verfahren. Symmetrie bedeutet, dass Entschlüsselungsschlüssel und Verschlüsselungsschlüssel leicht auseinander herleitbar sind, daher müssen beide Schlüssel auch geheim gehalten werden. Wollen zwei Teilnehmer eines Netzwerks geheim kommunizieren, so muss vorher über einen sicheren Kanal der Schlüsselaustausch stattfinden. Für n Teilnehmer im Netzwerk müssen $\binom{n}{2} = \frac{1}{2}n \cdot (n-1) \approx \frac{1}{2}n^2$ Schlüssel geheim übertragen und geschützt gespeichert werden. Die Schlüsselanzahl steigt also quadratisch mit der Anzahl der Teilnehmer. Derzeit gibt es circa $1,1 \cdot 10^9$ Internet-Nutzer, dies würde $6 \cdot 10^{17}$ Schlüssel erfordern, falls alle miteinander geheim kommunizieren wollen.



Die Verteilung und Verwaltung der Schlüssel ist problematisch bei hohen Nutzerzahlen.
Zum Anderen lassen sich mit Private-Key-Verfahren keine Signaturen ermöglichen. Signaturen dienen dazu, die Echtheit von Informationen zu bezeugen. Dies wird beispielsweise bei Finanz-Transaktionen im Internet (z.B. Abschließen von Kaufverträgen, Online-Banking) benötigt, die Private-Key-Verfahren bieten aber keine rechtliche Sicherheit. Es ist nämlich nicht möglich, gegenüber Dritten nachzuweisen, dass ein bestimmter Absender eine bestimmte Nachricht geschickt hat.
Diese beiden Probleme lösen die Public-Key-Verfahren. Jeder Teilnehmer hat zwei Schlüssel, einen zum Verschlüsseln und einen zum Entschlüsseln. Für n Teilnehmer im Netzwerk gibt es also $2n$ Schlüssel,

dabei müssen nur die n Entschlüsselungsschlüssel (geheimer Schlüssel, private key) geheimgehalten werden, die n Verschlüsselungsschlüssel (öffentlicher Schlüssel, public key) können veröffentlicht werden. Die Schlüsselanzahl steigt hier nur linear mit der Nutzerzahl.
Public-Key-Systeme sind asymmetrisch, d.h. der geheime Entschlüsselungsschlüssel kann aus dem öffentlichen Verschlüsselungsschlüssel nicht mit vertretbaren Aufwand ermittelt werden — ein Public-Key-Verfahren kann auch nur dann sicher sein.
Jeder Teilnehmer eines Public-Key-Netzwerks schreibt seinen öffentlichen Schlüssel in ein öffentliches Verzeichnis. Wollen nun zwei Teilnehmer geheim kommunizieren, so holt sich der Sender den öffentlichen Schlüssel des Empfängers, verschlüsselt mit ihm die Nachricht und verschickt sie. Der Empfänger entschlüsselt dann mit seinem privaten Schlüssel die Nachricht. Ein Vorteil ist demnach, dass kein (vorheriger) Schlüsselaustausch nötig und somit spontane Kommunikation möglich ist.



Das öffentliche Schlüsselverzeichnis muss aber vor Veränderungen geschützt werden (dies geschieht mit Signaturen), andernfalls könnte ein Angreifer den öffentlichen Verschlüsselungsschlüssel eines Teilnehmers X durch seinen eigenen ersetzen und damit die Nachrichten, die für X bestimmt sind, lesen.
Public-Key-Verfahren erleichtern aber nicht nur das Schlüsselmanagement erheblich, mit ihnen können auch Signaturen erzeugt werden, die die Authentizität von Informationen bezeugen können. Dies geschieht, indem ein Teilnehmer X einen Signaturtext mit seinem privaten Schlüssel verschlüsselt, und jeder andere Teilnehmer des Netzwerks kann dann mit dem öffentlichen Schlüssel von X entschlüsseln und damit die Signatur bestätigen.

Ein Public-Key-Verfahren zur Verschlüsselung & Signierung muss also vier Bedingungen erfüllen:

(K6)* Für jedes Schlüsselpaar ($\underbrace{e_X}_{\text{öffentl.}}$, $\underbrace{d_X}_{\text{privat}}$) von X muss

$$D_X \circ E_X = \mathrm{id}$$

gelten. Dabei ist E_X die zu e_X gehörige Verschlüsselungsfunktion und D_X die zu d_X gehörende Entschlüsselungsfunktion.

(E1)* Es gibt effiziente (in Speicherplatz und Rechenzeit) Berechnungsverfahren für E_X und D_X.

(E2)* d_X kann nicht aus e_X abgeleitet werden.
In der Praxis ersetzt man das Wort „nicht“ durch „mit den heute und auf absehbare Zeit zur Verfügung stehenden Möglichkeiten“.

(S)* Gilt außerdem für jedes Schlüsselpaar (e_X, d_X) von X

$$E_X \circ D_X = \mathrm{id},$$

so handelt es sich zusätzlich um ein Signaturverfahren. Die ersten drei Forderungen werden an reine Verschlüsselungsverfahren gestellt.

Die Forderungen (E1)* und (E2)* werden mit einer Einweg-Funktion mit Falltüre realisiert. Die Bedingung (K6)* finden wir später in der Definition 2.1 vom Kryptosystem, und auf (S)* bzw. auf Signaturen soll im Folgenden nicht weiter eingegangen werden.

Definition 1.1 (Einweg-Funktion mit Falltüre)
Eine injektive Funktion $f : X \longrightarrow Y$ heißt Einweg-Funktion mit Falltüre (trapdoor one-way function), falls folgendes gilt:

(E1) Es gibt effiziente Verfahren zur Berechnung von f und $f^{-1} : f(X) \longrightarrow X$.

(E2) Das effiziente Verfahren zur Berechnung von f^{-1} kann nicht aus f hergeleitet werden, sondern nur mit Hilfe einer geheim zu haltenden Zusatzinformation (Falltüre, trapdoor).

Grundlage für eine Einweg-Funktion mit Falltüre bilden bisher nicht gelöste Berechnungsprobleme der Zahlentheorie, z.B. Faktorisierung von ganzen Zahlen.
Die Existenz solcher Funktionen ist nicht beweisbar, es handelt sich also unter Umständen um eine unterhaltsame Definition der leeren Menge. Der Grund hierfür liegt darin, dass sich (fast) alle Funktionen theoretisch umkehren lassen, dies aber in der Praxis mit den zur Verfügung stehenden Mitteln nicht möglich ist. Im Allgemeinen scheitert eine Funktionsumkehrung dann am erforderlichen Aufwand an Rechenzeit und Speicherplatz.
Die Rechengeschwindigkeit sowie der Speicherplatz von Computern nimmt zwar gerade in letzter Zeit rasant zu, allerdings lassen sich gewisse physikalische Grenzen nicht überschreiten.

Beispiel
Sei n das Produkt zweier großer Primzahlen p und q, d.h. $n = p \cdot q > 10^{600}$, und sei $e \in \mathbb{N}$.
Die Funktion $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n : x \longmapsto x^e \mod n$ wird als eine Einweg-Funktion mit Falltüre angesehen, denn solange nur die Funktion selbst, also insbesondere e und n bekannt sind, existiert derzeit kein effizienter Algorithmus zur Umkehrung bzw. zur Bestimmung der e-ten Wurzel modulo n.
Die Falltüre ist in diesem Falle die Kenntnis der erzeugenden Primzahlen p und q, dann gibt es nämlich einen effizienten Algorithmus, der im Folgenden vorgestellt wird.

Schließlich möchte ich noch anmerken, dass Public-Key-Verfahren im Gegensatz zu den Private-Key-Verfahren sehr langsam sind. Daher verknüpft man beide miteinander, indem über ein schnelles Private-Key-Verfahren die eigentliche Kommunikation stattfindet und der vorherige Schlüsselaustausch über ein Public-Key-Verfahren geschieht, welches damit also den erforderlichen geheimen Kanal bereit stellt.

2 RSA-Verschlüsselung

Das heute noch wichtigste Public-Key-Verfahren, das RSA-Verschlüsselungsverfahren, ist benannt nach seinen Erfindern

Ronald **R***ivest, Adi* **S***hamir* und *Leonard* **A***dleman.*

Seine Sicherheit beruht auf dem ungelösten Faktorisierungsproblem: Es ist recht einfach, zwei Primzahlen p und q miteinander zu einer Zahl n zu multiplizieren, allerdings ist es ziemlich schwer, nur aus der Kenntnis der Zahl n auf die beiden Primfaktoren p und q zu schließen.

$$p \cdot q \underset{\longleftarrow}{\overset{\longrightarrow}{=}} n$$

Hierzu möchte ich folgendes Zitat anbringen:

> 1977 nahmen die drei Personen, die den spektakulärsten Einzelbeitrag zur Public-Key-Kryptographie leisten sollten, Ronald Rivest, Adi Shamir und Leonard Adleman, die Herausforderung an und produzierten ein alle Erwartungen erfüllendes Public-Key-Kryptosystem. Der Prozess dauerte mehrere Monate, während derer Rivest Vorschläge machte, Adleman sie angriff und Shamir sich erinnert, zu beidem beigetragen zu haben. Im Mai 1977 wurden sie mit Erfolg gekrönt. Sie hatten entdeckt, wie ein einfaches Stück klassischer Zahlentheorie benutzt werden kann, um das Problem zu lösen. (W. Diffie)

2.1 Das RSA-Kryptosystem

Zuerst soll die allgemeine Definition eines Kryptosystems vorgestellt werden.

Definition 2.1 (Kryptosystem)
Ein Kryptosystem ist ein Fünftupel $(\mathscr{P}, \mathscr{C}, \mathscr{K}, \mathscr{E}, \mathscr{D})$ mit folgenden Eigenschaften:

(K1) $\mathscr{P}$ ist eine Menge und heißt Klartextraum. Die Elemente heißen Klartexte.

(K2) $\mathscr{C}$ ist eine Menge und heißt Chiffretextraum. Die Elemente heißen Chiffretexte.

(K3) $\mathscr{K}$ ist eine Menge und heißt Schlüsselraum. Die Elemente heißen Schlüssel.

(K4) $\mathscr{E} = \{E_k : \mathscr{P} \longrightarrow \mathscr{C} \mid k \in \mathscr{K}\}$ ist eine Funktionenfamilie.
Die Elemente heißen Verschlüsselungsfunktionen.

(K5) $\mathscr{D} = \{D_k : \mathscr{C} \longrightarrow \mathscr{P} \mid k \in \mathscr{K}\}$ ist eine Funktionenfamilie.
Die Elemente heißen Entschlüsselungsfunktionen.

(K6) Für jedes $e \in \mathscr{K}$ gibt es ein $d \in \mathscr{K}$, sodass für alle $p \in \mathscr{P}$ die Gleichung $D_d\big(E_e(p)\big) = p$ erfüllt ist, d.h. zu jedem beliebig verschlüsselten Klartext existiert ein Entschlüsselungsschlüssel, sodass der Chiffretext korrekt zum ursprünglichen Klartext entschlüsselt wird.

($\mathscr{P}$...plaintext, $\mathscr{C}$...ciphertext, $\mathscr{K}$...key, $\mathscr{E}$...encryption, $\mathscr{D}$...decryption)

Nun definieren wir fünf Mengen, die zusammen als Fünftupel ein Kryptosystem bilden.

Definition 2.2 (RSA-Kryptosystem)
Sei $n = p \cdot q$ für zwei verschiedene Primzahlen p und q.
Klartextraum und Chiffretextraum sei $\mathscr{P} = \mathscr{C} = \mathbb{Z}_n$.
Als Schlüsselraum wird

$$\mathscr{K} = \left\{(n, p, q, d, e) \in \mathbb{N}^5 \;\middle|\; d \cdot e \equiv 1 \mod \varphi(n),\ 1 < d, e < \varphi(n)\right\}$$

gesetzt. Für einen Schlüssel $k \in \mathscr{K}$ werden die Verschlüsselungsfunktion

$$E_k : \mathscr{P} \longrightarrow \mathscr{C} : m \longmapsto m^e \mod n$$

und die Entschlüsselungsfunktion

$$D_k : \mathscr{C} \longrightarrow \mathscr{P} : c \longmapsto c^d \mod n$$

definiert.
$\left(\mathscr{P}, \mathscr{C}, \mathscr{K}, \mathscr{E}, \mathscr{D}\right)$ ist dann nach Satz 2.6 ein Kryptosystem, das sogenannte RSA-Kryptosystem.
Es handelt sich um ein Public-Key-Verfahren. Der öffentliche Schlüssel (public key) ist

$$(n, e),$$

der private Schlüssel (private key) ist

$$(p, q, d).$$

2.2 Beweis der Determiniertheit

Die Kryptosystem-Eigenschaften (K1) bis (K5) sind nach Definition 2.2 erfüllt, es ist demnach noch die Gültigkeit von (K6) zu zeigen. Dazu benötigen wir die folgenden beiden Sätze und die beiden Lemmata.

Satz 2.3 (Euler'sche φ-Funktion)
Für eine natürliche Zahl n definieren wir $\varphi(n)$ als die Anzahl der zu n teilerfremden natürlichen Zahlen, die nicht größer als n sind. Man nennt φ die Euler'sche φ-Funktion.

(1) Falls p eine Primzahl ist, so gilt

$$\varphi(p) = p - 1.$$

(2) Für zwei verschiedene Primzahlen p und q gilt

$$\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1).$$

Satz 2.4 (Fermat-Euler)
Sei m eine natürliche Zahl und p eine zu m teilerfremde Primzahl. Dann gilt

$$m^{\varphi(p)} \equiv 1 \mod p.$$

Das RSA-Verfahren ist eine Anwendung des Satzes von Fermat-Euler.

Lemma 2.5
Seien $a, b \in \mathbb{Z}$ und $n \in \mathbb{N}_+$. Dann gelten:

(1) $a \equiv b \mod n \implies \forall c \in \mathbb{Z} : a \cdot c \equiv b \cdot c \mod n$

(2) $a \equiv b \mod n \implies \forall i \in \mathbb{N} : a^i \equiv b^i \mod n$

Wir zeigen nun, dass das in Definition 2.2 definierte Fünftupel der Eigenschaft (K6) genügt, d.h. dass zu jeder Verschlüsselungsfunktion eine entsprechende Entschlüsselungsfunktion existiert.
Es ist zu zeigen, dass

$$\forall k_1 \in \mathcal{K} \; \exists k_2 \in \mathcal{K} : D_{k_2}\big(E_{k_1}(m)\big) = m$$

für alle $m \in \mathcal{P} = \mathbb{Z}_n$ gilt.
Es ergibt sich aus einem Schlüssel $k = (n, p, q, d, e) \in \mathcal{K}$ der öffentliche Schlüssel (n, e), der zum Verschlüsseln benutzt wird, sowie der private Schlüssel (p, q, d), der zum Entschlüsseln genommen wird. Es liegt also nahe, $k_1 = k_2 = k$ zu setzen. Mit

$$D_k\big(E_k(m)\big) = D_k(m^e \mod n) = (m^e \mod n)^d \mod n \overset{*}{=} (m^e)^d \mod n$$

haben wir demnach zu zeigen, dass

$$(m^e)^d \mod n = m$$

für alle $m \in \mathcal{P} = \mathbb{Z}_n$ erfüllt ist.
Die Gleichheit bei $*$ gilt, denn sei $m^e = k \cdot n + r$ für $k, r \in \mathbb{Z}$ mit $r < n$, dann gilt

$$k \cdot n + r \equiv r \mod n$$

und mit Lemma 2.5 (2) folgt

$$(k \cdot n + r)^d \equiv r^d \mod n.$$

Weiter ist $k \cdot n + r \mod n = r$ und damit

$$(k \cdot n + r)^d \equiv (k \cdot n + r \mod n)^d \mod n.$$

Mit Rücksubstitution und wegen der Symmetrie der modularen Äquivalenzrelation $\equiv_{\mod n}$ auf $\mathbb{Z}_n$ folgt

$$(m^e \mod n)^d \equiv (m^e)^d \mod n.$$

Satz 2.6
Sei (n, e) ein öffentlicher und (p, q, d) der entsprechende private Schlüssel im RSA-Verfahren. Dann gilt

$$(m^e)^d \mod n = m$$

für jeden Klartext $m \in \mathbb{Z}_n$.

Beweis
Es gilt $d \cdot e \equiv 1 \mod \varphi(n)$, also gibt es eine Zahl $k \in \mathbb{Z}$, sodass

$$d \cdot e = 1 + k \cdot \varphi(n)$$

ist. Es gilt sogar $k \in \mathbb{N}$, denn mit $1 < d, e$ gilt $1 < d \cdot e = 1 + k \cdot \varphi(n)$, also $0 < k \cdot \underbrace{\varphi(n)}_{>0}$.
Wegen $n = p \cdot q$ gilt $\varphi(n) = \varphi(p) \cdot \varphi(q)$. Demnach folgt für $m \in \mathbb{Z}_n$

$$(m^e)^d = m^{d \cdot e} = m^{1 + k \cdot \varphi(n)} = m \cdot (m^{\varphi(n)})^k = m \cdot (m^{\varphi(p) \cdot \varphi(q)})^k.$$

Nun betrachten wir zwei Fälle.

(a) $p \nmid m$
Ist p kein Teiler von m, d.h. $\gcd(m,p) = 1$, dann gilt nach dem Satz von Fermat-Euler

$$m^{\varphi(p)} \equiv 1 \mod p.$$

Potenzieren wir die Äquivalenz nach Lemma 2.5 (2) mit $\varphi(q) \cdot k \in \mathbb{N}$ und multiplizieren wir sie anschließend nach Lemma 2.5 (1) mit $m \in \mathbb{Z}_n \subset \mathbb{Z}$, so ergibt sich

$$m \cdot (m^{\varphi(p)})^{\varphi(q)\cdot k} \equiv m \cdot 1^{\varphi(q)\cdot k} \mod p$$

Also gilt dann

$$(m^e)^d \equiv m \mod p.$$

(b) $p \mid m$
Falls p jedoch ein Teiler von m ist, dann gilt

$$(m^e)^d \equiv m \equiv 0 \mod p.$$

Es gilt also stets $(m^e)^d \equiv m \mod p$. Mit analoger Argumentation folgt

$$(m^e)^d \equiv m \mod q.$$

Es gibt also Zahlen $s, t \in \mathbb{Z}$, sodass

$$(m^e)^d = m + s \cdot p \quad \text{und} \quad (m^e)^d = m + t \cdot q,$$

gelten, d.h. $s \cdot p = t \cdot q$.
Nun ist $\frac{s \cdot p}{q} = t \in \mathbb{Z}$, also ist s durch q teilbar, denn p ist prim. Es existiert ein $u \in \mathbb{Z}$, sodass $s = u \cdot q$.
Es gilt demzufolge

$$(m^e)^d \equiv m + u \cdot \underbrace{q \cdot p}_{=n} \equiv m \mod n$$

und mit $m \in \mathbb{Z}_n$, d.h. $0 \leqslant m < n$ folgt die Behauptung. □

Analog gilt $(s^d)^e \mod n = s$ für alle Signaturtexte $s \in \mathbb{Z}_n$, das RSA-Verfahren eignet sich also auch als Signaturverfahren.

2.3 Schlüsselerzeugung

Algorithmus 2.7

(1) Wähle zwei „große“ Primzahlen p und q mit $p \neq q$.

(2) Setze $n = p \cdot q$ und $\varphi(n) = (p-1) \cdot (q-1)$.

(3) Wähle ein beliebiges e mit $1 < e < \varphi(n)$ und $\gcd(e, \varphi(n)) = 1$.

(4) Berechne ein d mit $1 < d < \varphi(n)$ und $d \cdot e \equiv 1 \mod \varphi(n)$.

Öffentlicher Schlüssel: (n, e)
Privater Schlüssel: (p, q, d)
(n…RSA-Modul, e…Verschlüsselungsexponent, d…Entschlüsselungsexponent)

Die Primzahlen können mit dem *„Algorithmus für die zufällige Wahl von Primzahlen“* A.3, der auf dem Miller-Rabin-Test aufbaut, erzeugt werden. Für den Verschlüsselungsexponenten e kommen nur ungerade Zahlen in Frage, denn $\varphi(n) = (p-1)\cdot(q-1)$ ist stets gerade und es soll $\gcd(e,\varphi(n)) = 1$ gelten.
Das Lösen der Kongruenz $d \cdot e \equiv 1 \mod \varphi(n)$ erfolgt mit dem erweiterten euklidischen Algorithmus.
Es gilt

$$1 = \gcd(\varphi(n), e),$$

der erweiterte euklidische Algorithmus liefert uns Zahlen $k, d \in \mathbb{Z}$ mit

$$1 = k \cdot \varphi(n) + d \cdot e.$$

Es wird aber nur d benötigt, denn dann gilt

$$k \cdot \varphi(n) + d \cdot e \equiv d \cdot e \equiv 1 \mod \varphi(n).$$

Das gesuchte d ist das modulare Inverse zu e und kann mit dem *„Algorithmus zur modularen Invertierung“* A.5 bestimmt werden.
Dieses d ist eindeutig bestimmt, denn sei $d \in \mathbb{Z}$ mit $d' \neq d$ sowie $1 < d' < \varphi(n)$ und $d' \cdot e \equiv 1 \mod \varphi(n)$, dann wäre

$$d' \cdot e \equiv d \cdot e \mod \varphi(n).$$

Wegen $\gcd(e,\varphi(n) = 1$ ließe sich diese Äquivalenz kürzen zu

$$d' \equiv d \mod \varphi(n),$$

und mit $1 < d, d' < \varphi(n) < n$ fölge nun der Widerspruch $d' = d$.
Es ist ebenso möglich, in (3) das d zu wählen und dann in (4) das e zu berechnen.

Beispiel 1

Wir wählen die Primzahlen $p = 3$ und $q = 5$.
Das RSA-Modul ist dann $n = p \cdot q = 15$,
$\varphi(n) = (p-1)\cdot(q-1) = 8$.
Wir wählen $e = 3$.

i	0	1	2	3	4
r_i	8	3	2	1	0
q_i	/	2	1	2	/
d_i	0	1	2	3	8

Der Algorithmus liefert also $d = 3$.
Damit haben wir den Schlüssel
$(n, p, q, d, e) = (15, 3, 5, 3, 3)$
und damit den öffentlichen Schlüssel

$$(15, 3)$$

sowie den privaten Schlüssel

$$(3, 5, 3).$$

Beispiel 2

Wir wählen die Primzahlen $p = 101$ und $q = 107$.
Das RSA-Modul ist dann $n = p \cdot q = 10807$,
$\varphi(n) = (p-1)\cdot(q-1) = 10600$.
Wir wählen $e = 523$.

i	0	1	2	3	4	5	6	7	8	9	10	11
r_i	10600	523	140	103	37	29	8	5	3	2	1	0
q_i	/	20	3	1	2	1	3	1	1	1	2	/
d_i	0	1	20	61	81	223	304	1135	1439	2574	4013	10600

Der Algorithmus liefert also $d = -4013$,
wir addieren $\varphi(n) = 10600$ und erhalten $d' = 6587$.
Damit haben wir den Schlüssel
$(n, p, q, d, e) = (10807, 101, 107, 6587, 523)$
und damit den öffentlichen Schlüssel

$$(10807, 523)$$

sowie den privaten Schlüssel

$$(101, 107, 6587).$$

2.4 Verschlüsselung & Entschlüsselung

Ein Klartext $m \in \mathscr{P} = \mathbb{Z}_n$ wird mit dem öffentlichen Schlüssel (n, e) verschlüsselt zu

$$c = m^e \mod n.$$

Die zugehörige Verschlüsselungsfunktion ist $E_{(n,e)}(m) = m^e \mod n$.
Die Berechnung kann mittels dem *„Algorithmus zur schnellen Exponentiation"* A.6 erfolgen.

Beispiel 1
Wir haben den öffentlichen Schlüssel $(n, e) = (15, 3)$, die zugehörige Verschlüsselungsfunktion ist damit $E_{(15,3)}(m) = m^3 \mod 15$.
Nun verschlüsseln wir den Klartext $m = 7$,
es ist $c = E_{(15,3)}(7) = 7^3 \mod 15 = 13$.

Beispiel 2
Wir haben den öffentlichen Schlüssel $(n, e) = (10807, 523)$, also die Verschlüsselungsfunktion $E_{(10807,523)}(m) = m^{523} \mod 10807$.
Nun verschlüsseln wir den Klartext $m = 7653$ zu
$c = E_{(10807,523)}(7653) = 7653^{523} \mod 10807 = 8968$.

Ein Chiffretext $c \in \mathscr{C} = \mathbb{Z}_n$ wird mit dem privaten Schlüssel (p, q, d) entschlüsselt zu

$$m = c^d \mod n.$$

Die entsprechende Entschlüsselungsfunktion ist $D_{(p,q,d)}(c) = c^d \mod n$ mit $n = p \cdot q$.
Die Berechnung kann auch hier mit dem *„Algorithmus zur schnellen Exponentiation"* A.6 erfolgen.

Beispiel 1
Wir haben den privaten Schlüssel $(p, q, d) = (3, 5, 3)$, die zugehörige Entschlüsselungsfunktion ist damit $D_{(3,5,3)}(c) = c^3 \mod 15$.
Nun entschlüsseln wir den Chiffretext $c = 13$,
es ist $m = D_{(3,5,3)}(13) = 13^3 \mod 15 = 7$.

Beispiel 2
Wir haben den öffentlichen Schlüssel $(p, q, d) = (101, 107, 6587)$ mit der Entschlüsselungsfunktion $D_{(101,107,6587)}(c) = c^{6587} \mod 10807$.
Nun entschlüsseln wir den Chiffretext $c = 8968$:
$m = D_{(101,107,6587)}(8968) = 8968^{6587} \mod 10807 = 7653$.

2.5 RSA-Blockchiffre

Wir haben gesehen, wie mit dem RSA-Verfahren Zahlen, die kleiner als das RSA-Modul sind, verschlüsselt werden können. Ein Klartext $m \in \mathscr{P} = \mathbb{Z}_n$ wird mit dem öffentlichen Schlüssel (n,e) verschlüsselt zu $c = E_{(n,e)}(m) = m^e \mod n$.
Im Folgenden soll gezeigt werden, wie mittels RSA auch Texte verschlüsselt werden können.
Dazu sei Σ das verwendete Alphabet mit $|\Sigma| = N$, das Alphabet habe also N Symbole. Weiter gebe es eine bijektive Abbildung $\Sigma \longrightarrow \mathbb{Z}_N$, die jedem Buchstaben eine Zahl aus $\{0,1,\ldots,N-1\}$ eineindeutig zuordne. Jeder Buchstabe stellt also genau eine Zahl dar und im Folgenden wird mit diesen zugeordneten Zahlen gerechnet. Wir setzen

$$k = \lfloor \log_N n \rfloor.$$

Ein Klartextblock $m_1 \ldots m_k$ mit $m_i \in \Sigma$ für $i \in \{1,\ldots,k\}$ wird in die Zahl

$$m = \sum_{i=1}^{k} m_i N^{k-i}$$

umgewandelt. Der Block $m_1 \ldots m_k$ ist also die N-adische Entwicklung von m. Es ist ausgeschlossen, dass dabei eine Zahl m entsteht, die größer als das RSA-Modul n ist, denn wegen $k = \lfloor \log_N n \rfloor$ gilt

$$0 \leqslant m \leqslant (N-1) \cdot \sum_{i=1}^{k} N^{k-i} = N^k - 1 < n.$$

Nun können wir m zu $c = E_{(n,e)}(m) = m^e \mod n$ verschlüsseln.
Die Zahl c kann dann wieder zur Basis N geschrieben werden. Unter Umständen hat allerdings die N-adische Entwicklung von c die Länge $k+1$. Es ist nämlich möglich, dass m zu einem c mit $N^k - 1 < c < n$ verschlüsselt wird. Dann ist $c \geqslant N^k$ und wir benötigen demzufolge eine weitere Stelle $c_0 \neq 0$ zur Darstellung. Wir schreiben also

$$c = \sum_{i=0}^{k} c_i N^{k-i} \quad \text{mit } c_i \in \Sigma \text{ für } i \in \{0,\ldots,k\}.$$

Wir erhalten somit den Chiffretextblock $c_0 c_1 \ldots c_k$.
Mit diesem RSA-Blockverfahren werden Klartextblöcke der Länge k injektiv auf Chiffretextblöcke der Länge $k+1$ abgebildet. Es ist also keine Blockchiffre im eigentlichen Sinn, nämlich dass der Klartextraum und der Chiffretextraum die Menge Σ^k aller Wörter der Länge k über einem Alphabet Σ sind, sondern hier ist der Klartextraum Σ^k und der Chiffretextraum Σ^{k+1}. Mit kleinen Modifikationen kann mit der hier vorgestellten RSA-Blockversion der ECB-Mode und der CBC-Mode angewendet werden. Es jedoch nicht ratsam, den CFB-Mode oder OFB-Mode zu benutzen, denn beide nutzen nur die Verschlüsselungsfunktion. Die Verschlüsselungsfunktion ergibt sich aus dem öffentlichen Schlüssel, also würde sie auch jeder Angreifer kennen.
Möchte man die Sicherheit dieser RSA-Blockchiffre steigern, so kann man jeden Klartext mehrfach hintereinander mit verschiedenen Schlüsseln verschlüsseln.

2.6 ASCII-Blockchiffre im ECB-Mode

Es soll nun eine RSA-Blockchiffre für ASCII-Texte im ECB-Mode vorgestellt werden. Das verwendete ASCII-Alphabet besteht aus 256 Zeichen, die durch Binärblöcke der Bit-Länge 8 dargestellt werden. Wir setzen also

$$k = \lfloor \log_{256} n \rfloor = \lfloor \frac{1}{8} \cdot \log_2 n \rfloor$$

für den gegebenen öffentlichen Schlüssel (n, e). Dieses k gibt uns die Länge der Blöcke vor.
Um einen ASCII-Text zu verschlüsseln, könnte man nun folgendermaßen vorgehen:

(1) Unterteilung des zu verschlüsselnden Textes in Blöcke der Länge k (bei 513 bit-Modul also 64-Zeichen-Blöcke)

(2) Ersetzung der Buchstaben durch ihre jeweilige ASCII-Binärdarstellung, dabei entstehen Binärblöcke der Länge $8k$ (bei 64 Zeichen also 512 bit-Blöcke) [falls der letzte Block nicht die Länge k bzw. $8k$ hat, so wird mit genügend Binärblöcken „00000000“ aufgefüllt]

(3) Interpretation der entstandenen Blöcke als binäre Darstellung von natürlichen Zahlen m_i

(4) Verschlüsselung der m_i zu c_i

(5) Übergabe als Block $c_1 \dots c_k$

Die Entschlüsselung eines derartig erzeugten Blocks $c_1 \dots c_k$ geschieht dann so:

(1) Entschlüsselung der c_i zu m_i

(2) Ermittlung der binären Darstellung der m_i (Bit-Länge muss durch 8 teilbar sein)

(3) Konkatenation der Binärblöcke und anschließend Aufteilung in Blöcke der Länge 8

(4) Ersetzen der 8 bit-Binärblöcke durch den jeweils zugeordneten Buchstaben des ASCII-Alphabets [Binärblöcke „00000000“ können ignoriert werden]

!	00100001	A	01000001	a	01100001
“	00100010	B	01000010	b	01100010
#	00100011	C	01000011	c	01100011
$	00100100	D	01000100	d	01100100
%	00100101	E	01000101	e	01100101
&	00100110	F	01000110	f	01100110
’	00100111	G	01000111	g	01100111
(	00101000	H	01001000	h	01101000
)	00101001	I	01001001	i	01101001
,	00101100	J	01001010	j	01101010
-	00101101	K	01001011	k	01101011
.	00101110	L	01001100	l	01101100
/	00101111	M	01001101	m	01101101
:	00111010	N	01001110	n	01101110
;	00111011	O	01001111	o	01101111
?	00111111	P	01010000	p	01110000
0	00110000	Q	01010001	q	01110001
1	00110001	R	01010010	r	01110010
2	00110010	S	01010011	s	01110011
3	00110011	T	01010100	t	01110100
4	00110100	U	01010101	u	01110101
5	00110101	V	01010110	v	01110110
6	00110110	W	01010111	w	01110111
7	00110111	X	01011000	x	01111000
8	00111000	Y	01011001	y	01111001
9	00111001	Z	01011010	z	01111010

Einige ASCII-Zeichen und deren binäre Darstellung

Schließlich entsteht auf diesem Weg wieder der anfangs verschlüsselte Klartext.

Für ein 512 bit-Modul n gilt $2^{511} \leqslant n < 2^{512}$, also ist $k = \lfloor \frac{1}{8} \cdot \log_2 n \rfloor < 64$.
Mit kleinen Änderungen kann auch ein 512 bit-Modul verwendet werden, sodass die Blocklänge 64 gewählt werden kann. Dies impliziert folgendes Problem:
Steht an erster Stelle eines Blocks ein Sonderzeichen mit 1 als erstes Bit der binären Darstellung, dann ist die durch den Block binär dargestellte natürliche Zahl unter Umständen größer als das RSA-Modul und kann somit nicht verschlüsselt werden.
In so einem Fall könnte man das Sonderzeichen und alle folgenden Zeichen eine Position nach hinten schieben, und den Block „00000000“ einfügen. Damit ist der Block dann kleiner als das RSA-Modul und kann verschlüsselt werden. Bei der Entschlüsselung muss dann ein Binärblock „00000000“ nicht weiter beachtet werden.
Eine andere Möglichkeit wäre, den RSA-Modul ein Bit länger zu wählen, also 513 bit wie oben.

Auf ähnliche Art und Weise könnte auch eine Unicode-Blockchiffre realisiert werden. Im Unicode-Alphabet werden die Symbole durch 16 bit-Binärblöcke dargestellt, es hat demnach 65536 Zeichen.

Beispiel

Der Text „RSA-Verschluesselung“ soll verschlüsselt werden.

Wir wollen den Text in Blöcke mit je 2 Buchstaben aufteilen, wir benötigen also eigentlich ein 17 bit-Modul. Es soll aber die oben vorgestellte Modifikation benutzt werden, sodass wir ein 16 bit-Modul benutzen können. Wir wählen die beiden 8 bit-Primzahlen $p = 233$ und $q = 239$.

Das RSA-Modul ist also $n = 55687_{(10)} = 1101100110000111_{(2)}$ und $\varphi(n) = 55216$. Wir wählen $e = 11$.

i	0	1	2	3	4	5	6
r_i	55216	11	7	4	3	1	0
q_i	/	5019	1	1	1	3	/
d_i	0	1	5019	5020	10039	15059	55216

Der Algorithmus liefert also $d = 15059$.

Damit haben wir folgenden Schlüssel erzeugt:

- Öffentlicher Schlüssel: $(n, e) = (55687, 11)$
- Privater Schlüssel: $(p, q, d) = (233, 239, 15059)$

Text-Blöcke: RS A- Ve rs ch lu es se lu ng

Umgewandelter Text:

01010010 01010011 01000001 00101101 01010110 01100101 01110010 01110011 01100011 01101000 01101100 01110101 01100101 01110011 01110011 01100101 01101100 01110101 01101110 01100111

Der erste Block $0101001001010011_{(2)}$ entspricht der Zahl $m_1 = 21075_{(10)}$.

Diese wird verschlüsselt zu $c_1 = m_1^e \mod n = 21075^{11} \mod 55687 = 50989$.

Entschlüsselung liefert $m_1 = c_1^d \mod n = 50989^{15059} \mod 55687 = 21075$.

Die Binärdarstellung der Zahl ist $\underbrace{01010010}_{\text{R}}\underbrace{01010011}_{\text{S}}$.

Die folgende Tabelle stellt die Verschlüsselung sowie die Entschlüsselung dar:

i	Block	binäre Darstellung	Zahl m_i	$c_i = m_i^e \mod n$	$m_i = c_i^d \mod n$
1	RS	0101001001010011	21075	50989	21075
2	A-	0100000100101101	16685	15996	16685
3	Ve	0101011001100101	22117	42415	22117
4	rs	0111001001110011	29299	30568	29299
5	ch	0110001101101000	25448	38220	25448
6	lu	0110110001110101	27765	9151	27765
7	es	0110010101110011	25971	26285	25971
8	se	0111001101100101	29541	50383	29541
9	lu	0110110001110101	27765	9151	27765
10	ng	0110111001100111	28263	49236	28263

Der Text „RSA-Verschluesselung“ wird also blockweise verschlüsselt

zu dem Tupel (50989,15996,42415,30568,38220,9151,26285,50383,9151,49236).

Dieses Tupel wird schließlich wieder zu „RSA-Verschluesselung“ entschlüsselt.

Beispielsweise ist hier der Block „ÿÿ“, der binär als „11111111 11111111“ dargestellt wird, nicht verschlüsselbar, denn $1111111111111111_{(2)} = 65535_{(10)}$ ist größer als das RSA-Modul.

2.7 Sicherheit & Angriffe

2.7.1 Auswahl von d

◇ Es sollte gelten $d \geqslant n^{0,292}$.
Boneh & Durfee haben bewiesen, dass das Verfahren andernfalls gebrochen werden kann.

2.7.2 Faktorisierung (Auswahl von n, p, q)

Das Problem, den RSA-Modul zu faktorisieren (Faktorisierungsproblem), und das Problem, den Entschlüsselungsexponenten aus dem öffentlichen Schlüssel herzuleiten (RSA-Problem), sind äquivalent. Wird also in Zukunft eins der beiden Probleme gelöst, so ist es automatisch auch das andere.
Es muss sichergestellt werden, dass das erzeugte RSA-Modul n nicht von bekannten Faktorisierungsmethoden zerlegt werden kann. Gelingt es nämlich einem Angreifer, das RSA-Modul n zu faktorisieren und damit die beiden erzeugenden Primfaktoren p und q zu finden, so kann er mit der Kenntnis des Verschlüsselungsexponenten e aus dem öffentlichen Schlüssel leicht den Entschlüsselungsexponenten d berechnen, indem er den *„Algorithmus zur RSA-Schlüsselerzeugung“* 2.5 anwendet.

◇ Um eine Faktorisierung mit dem *Quadratischen Sieb* und dem *Zahlkörpersieb* nahezu unmöglich zu machen, sollte das RSA-Modul n mindestens $768\,bit$ lang sein, für höhere und längerfristige Sicherheit $1024\,bit$ oder sogar $2048\,bit$. Die binäre Länge einer natürlichen Zahl n ergibt sich aus $\text{size}(n) = \lfloor \log_2 n \rfloor + 1$. Für ein $768\,bit$-Modul beispielsweise gilt demnach $2^{767} \leqslant n < 2^{768}$.

◇ Die erzeugenden Primzahlen p und q sollten etwa gleich groß (mit der gleichen Bitlänge) gewählt werden, d.h. für ein $768\,bit$-Modul sollten beide Primfaktoren etwa $384\,bit$ lang sein. Dies verhindert die *Elliptische-Kurven-Methode (ECM)*.
Derart große Primfaktoren des Moduls erschweren auch die *Probedivision* erheblich. Andernfalls könnte man nämlich mit kleinen Primzahlen aus der Umgebung der 1 oder mit großen Primzahlen aus der Umgebung von n probeweise dividieren.

◇ Die Differenz $p - q$ sollte nicht zu klein sein, andernfalls wäre $p \approx q$ und somit $p \approx \sqrt{n}$. Dann könnte nämlich n einfach durch *Probedivision* aller Primzahlen aus der Umgebung von $\sqrt{n}$ faktorisiert werden.

◇ In einem Netzwerk sollten alle Teilnehmer paarweise verschiedene RSA-Moduln haben. Ansonsten könnte ein Teilnehmer aus der Kenntnis der Faktorisierung seines Moduls die privaten Schlüssel aller Teilnehmer mit demselben RSA-Modul ermitteln.

◇ Zur weiteren Erhöhung der Sicherheit sollten p und q starke Primzahlen sein, d.h. es sollen gelten

(a) $p - 1$ hat einen großen Primfaktor r,

(b) $p + 1$ hat einen großen Primfaktor, und

(c) $r - 1$ hat einen großen Primfaktor.

Die Forderung (a) verhindert die $p - 1$-*Methode*, (b) lässt die $p + 1$-*Methode* nicht zu und (c) stellt sicher, dass die *iterative Attacke* nicht zum Ziel führt.

2.7.3 Iterative Attacke

Sei $c = m^e \mod n$ ein Chiffretext. Wir setzen $k \in \mathbb{N}_+$ als kleinste Zahl, sodass

$$c^{e^k} \equiv c \mod n$$

gilt. Nun lässt sich folgende Zahlenfolge in $\mathbb{Z}_n$ definieren:
Es seien $c_0 = m$ und $c_1 = c = m^e \mod n$ die Startglieder und es gelte die Rekursionsformel

$$c_{i+1} = c_i^e \mod n \quad (k \in \mathbb{N}).$$

Dies ist genau die Verschlüsselungsfunktion, die auch jeder Angreifer kennt. Diese Funktion ist eine Selbstabbildung des $\mathbb{Z}_n$ und bijektiv, also eine Permutation auf $\mathbb{Z}_n$. Demnach muss man bei einer genügend großen Anzahl k an Hintereinanderausführungen wieder auf den ursprünglichen Chiffretext kommen – ein solches k existiert also.
Nun ist die Folge $(c_i)_{i\in\mathbb{N}}$ zyklisch mit der Länge k, damit erhält man also den Klartext mit

$$c^{e^{k-1}} \equiv m \mod n = m.$$

Dies sind die Grundgedanken dieses Angriffs, die natürlich noch weiter ausgebaut werden können.

2.7.4 Low-Exponent-Attacke (Auswahl von e)

◇ Die Verwendung von kleinen Verschlüsselungsexponenten ist nicht ungefährlich, daher sollten kleine Verschlüsselungsexponenten e, z.B. $e = 3, 5, 7, \ldots$ vermieden werden.

Unter gewissen Umständen kann ein Angreifer nämlich mit einer Low-Exponent-Attacke das RSA-Kryptosystem angreifen, die auf folgendem Satz beruht.

Satz 2.8
Seien $e \in \mathbb{N}$, $n_1, n_2, \ldots, n_e \in \mathbb{N}$ paarweise teilerfremd und $m \in \mathbb{N}$ mit $0 \leqslant m < n_i$ für alle $i \in \{1, \ldots, e\}$.
Sei weiter $c \in \mathbb{N}$ mit $c \equiv m^e \mod n_i$ für alle $i \in \{1, \ldots, e\}$ und $0 \leqslant c < \prod_{i=1}^{e} n_i$.
Dann folgt $c = m^e$.

Beweis
Die Zahl $c' = m^e$ erfüllt die simultane Kongruenz $c' \equiv m^e \mod n_i$ für alle $i \in \{1, \ldots, e\}$ und es gilt $0 \leqslant c' < \prod_{i=1}^{e} n_i$, weil $0 \leqslant m < n_i$ für alle $i \in \{1, \ldots, e\}$ vorrausgesetzt ist. Eine solche Lösung der simultanen Kongruenz ist aber nach dem chinesischen Restsatz eindeutig bestimmt, also folgt $c = c'$. □

Die Low-Exponent-Attacke kann immer dann angewendet werden, falls ein Klartext m mit e paarweise teilerfremden RSA-Moduln aber stets mit demselben Verschlüsselungsexponenten e verschlüsselt wird. Ein Angreifer kann dann den Klartext m folgendermaßen berechnen:

(1) Bestimmung einer Zahl c mit $c \equiv c_i \mod n_i$ für alle $i \in \{1, \ldots, e\}$ und $0 \leqslant c < \prod_{i=1}^{e} n_i$ mit dem chinesischen Restsatz.

(2) Nach obigen Satz ist $c = m^e$, also ergibt sich $m = \sqrt[e]{c}$.

Ohne jegliche Kenntnis eines Entschlüsselungsexponenten kann der Klartext damit rekonstruiert werden.

Beispiel

Wir wollen den Klartext $m = 12$ mit dem Verschlüsselungsexponenten $e = 3$ und drei paarweise teilerfremden RSA-Moduln verschlüsseln.
Dazu erzeugen wir zuerst drei geeignete RSA-Moduln, siehe nebenstehende Tabelle.

i	p_i	q_i	n_i	$\varphi(n_i)$	e
1	3	11	33	20	3
2	5	17	85	64	3
3	23	29	667	616	3

public key 1: (33,3)
$c_1 = 12^3 \mod 33 = 12$

public key 2: (85,3)
$c_2 = 12^3 \mod 85 = 28$

public key 3: (667,3)
$c_3 = 12^3 \mod 667 = 394$

Wir haben also die simultane Kongruenz

$$c \equiv 12 \mod 33, \quad c \equiv 28 \mod 85, \quad c \equiv 394 \mod 667$$

zu lösen, dies geschieht mit dem chinesischen Restsatz.
Wir setzen $n := \prod_{i=1}^{3} n_i$ und $N_i := \frac{n}{n_i}$ für $i \in \{1,2,3\}$, es sind also

$$n = 1870935 \quad \text{und} \quad N_1 = 56695,\ N_2 = 22011,\ N_3 = 2805.$$

Es gilt nun

$$\gcd(N_i, n_i) = 1 \quad \text{für } i \in \{1,2,3\},$$

denn die RSA-Moduln sind paarweise teilerfremd. Nun lassen sich mit dem erweiterten euklidischen Algorithmus Zahlen x_i und y_i berechnen, sodass $1 = x_i \cdot N_i + y_i \cdot n_i$ für $i \in \{1,2,3\}$ gilt. Dann ist

$$x_i \cdot N_i \equiv 1 \mod n_i \quad \text{für } i \in \{1,2,3\},$$

wir benötigen also nur die Koeffizienten x_i.

k	0	1	2	3
r_k	56695	33	1	0
q_k	/	1718	33	/
$x_{(k)}$	1	0	1	33

Wir erhalten $x_1 = 1$.

k	0	1	2	3	4	5
r_k	22011	85	81	4	1	0
q_k	/	258	1	20	4	/
$x_{(k)}$	1	0	1	1	21	85

Es ergibt sich $x_2 = 21$.

k	0	1	2	3	4	5	6	7	8	9	10
r_k	2805	667	137	119	18	11	7	4	3	1	0
q_k	/	4	4	1	6	1	1	1	1	3	/
$x_{(k)}$	1	0	1	4	5	34	39	73	112	185	667

Der Algorithmus liefert $x_3 = -185$.
Nach dem chinesischen Restsatz ist dann die Lösung

$$c = \sum_{i=1}^{3} c_i \cdot x_i \cdot N_i \mod n,$$

es ergibt sich demnach $c = -190833642 \mod 1870935 = 1728$. Nun ist nach Satz 2.8 $c = m^e$, also folgt

$$m = \sqrt[3]{1728} = 12.$$

Der Angriff ist also erfolgreich.

2.8 Effizienz

Beschleunigung der Verschlüsselung

Die Verschlüsselung erfordert eine Exponentiation modulo n. Die Verschlüsselung kann beschleunigt werden, indem kleine Exponenten e oder Exponenten mit einer geringen Anzahl an Einsen in der Binärdarstellung gewählt werden.

Wählt man z.B. $e = 2^{16}+1 = 65537_{(10)} = 10000000000000001_{(2)}$, dann werden bei der Berechnung mittels schneller Exponentiation 16 modulare Quadrierungen und nur eine modulare Multiplikation durchgeführt.

Beschleunigung der Entschlüsselung

Das Entschlüsseln kann um den Faktor 4 beschleunigt werden, wenn man den chinesischen Restsatz benutzt. Dazu berechnet man

$$m_p = c^d \mod p \quad \text{und} \quad m_q = c^d \mod q$$

und löst dann mit dem chinesischen Restsatz die simultane Kongruenz

$$m \equiv m_p \mod p \quad \text{und} \quad m \equiv m_q \mod q.$$

Da $\gcd(p,q) = 1$ ist, kann man mit dem erweiterten euklidischen Algorithmus Zahlen y_p und y_q berechnen mit

$$y_p \cdot p + y_q \cdot q = 1.$$

Dann ist schließlich

$$m = m_p \cdot y_q \cdot q + m_q \cdot y_p \cdot p \mod n$$

der gesuchte ursprüngliche Klartext.

Man beachte, dass die Zahlen $y_p \cdot p \mod n$ und $y_q \cdot q \mod n$ nicht von der zu entschlüsselnden Nachricht abhängen, und daher ein für allemal vorberechnet werden können.

A Algorithmen

A.1 Primzahlerzeugung

Die Grundlage des Miller-Rabin-Testes ist folgender Satz, der eine „Verschärfung“ des kleinen Satzes von Fermat ist. Wir setzen

$$n-1 = 2^s \cdot d \quad \text{für } s = \max\{r \in \mathbb{N} : 2^r \mid n-1\}.$$

Das heißt 2^s ist die größte Potenz von 2, die $n-1$ teilt.

Satz A.1
Ist n eine Primzahl und ist a eine zu n teilerfremde ganze Zahl, so gilt unter der obigen Festlegung entweder

$$a^d \equiv 1 \mod n \qquad *$$

oder es gibt ein $r \in \{0,1,\ldots,s-1\}$ mit

$$a^{2^r d} \equiv -1 \mod n. \qquad **$$

Algorithmus A.2 (Primzahltest, Miller-Rabin-Test)
Der folgende Algorithmus überprüft, ob eine gegebene ungerade Zahl n prim ist.

(1) Wähle zufällig und gleichverteilt eine Basis a mit $a \in \{2,...,n-1\}$.

(2) Wenn $\gcd(a,n) > 1$, dann ist n zusammengesetzt ⇒ Abbruch.

(3) Berechne $a^d, a^{2d}, ..., a^{2^{s-1}d}$ und prüfe ob $*$ oder $**$ erfüllt werden.
(Aus a^d erhält man alle folgenden Potenzen durch Quadrierungen.)

- wenn ja, dann ist n mit einer Wahrscheinlichkeit von $\frac{3}{4}$ prim.
- wenn nein, dann ist n zusammengesetzt ⇒ Abbruch.

(4) Beginne wieder bei (1) mit einer anderen Zahl a und führe den Algorithmus $t-$mal aus.

Algorithmus A.3 (Zufällige Wahl von Primzahlen)

(1) Bestimme $k-2$ Bits b_i zufällig und gleichverteilt mit $i \in \{1,..,k-2\}$.

(2) Betrachte $n = 1 b_{k-2} ... b_1 1$.

(3) Führe eine Probedivision durch, prüfe dabei ob n durch eine Primzahl p mit $p < 10^6$ teilbar ist. (Die Primzahlen p stehen in einer Liste und wurden vorher vielleicht mit den Sieb des Eratothenes bestimmt. Die Probedivision ist sinnvoll, da rund 95% aller zusammengesetzten Zahlen hierbei aussortiert werden.)

- Teiler gefunden ⇒ Gehe zu (1).

(4) Wende den Miller-Rabin-Test t-mal an.

- n besteht den Test nicht ⇒ Gehe zu (1).
- n besteht den Test ⇒ n ist mit hoher Warscheinlichkeit prim.

A.2 Berechnung des modularen Inversen

Satz A.4
Sind x und n teilerfremde ganze Zahlen, so gibt es eine ganze Zahl y mit

$$x \cdot y \equiv 1 \mod n.$$

x ist dann invertierbar modulo n und y heißt modulares Inverses zu x.

Zum Lösen der Kongruenz $d \cdot e \equiv 1 \mod \varphi(n)$ bestimmt man also das modulare Inverse zu e, dies kann folgendermaßen geschehen:

Algorithmus A.5 (Modulare Invertierung)
Es gilt

$$1 = \gcd(\varphi(n), e),$$

der erweiterte euklidische Algorithmus liefert uns Zahlen $k, d \in \mathbb{Z}$ mit

$$1 = k \cdot \varphi(n) + d \cdot e.$$

Es wird aber nur d benötigt, denn dann gilt

$$k \cdot \varphi(n) + d \cdot e \equiv d \cdot e \equiv 1 \mod \varphi(n).$$

Dies gibt Anlass, den erweiterten euklidischen Algorithmus für unseren Zweck etwas zu modifizieren:

i	0	1	2	…	m	$m+1$
r_i	$\varphi(n)$	e	…	…	1	0
q_i	⁄	…	…	…	…	⁄
d_i	0	1	…	…	d_m	$\varphi(n)$

Anfangswerte: $r_0 = \varphi(n),\ r_1 = e,\ d_0 = 0,\ d_1 = 1$
Iteration: $r_{i+1} = r_{i-1} - q_i \cdot r_i$
$q_{i+1} = \left\lfloor \frac{r_i}{r_{i+1}} \right\rfloor$
$d_{i+1} = d_{i-1} + q_i \cdot d_i$

Wir erhalten damit ein d' mittels

$$d' = (-1)^{m+1} \cdot d_m$$

und es gilt dann $d' \cdot e \equiv 1 \mod \varphi(n)$. Es ist allerdings nach obiger Formel nicht auszuschließen, dass wir ein negatives d' erhalten. Es gilt

$$|d'| \leqslant \frac{1}{2}\varphi(n),$$

also addiert man in dem Fall einfach $\varphi(n)$ zu d', d.h.

$$d^* = d' + \varphi(n),$$

sodass dann $1 < d^* < \varphi(n)$ gilt. Das d^* erfüllt dann auch die Kongruenz

$$d^* \cdot e \equiv \left(d' + \varphi(n)\right) \cdot e \equiv d' \cdot e + e \cdot \varphi(n) \equiv d' \cdot e \equiv 1 \mod \varphi(n).$$

Schließlich ist das gesuchte d also

$$d = \begin{cases} d' & d' > 0 \\ d' + \varphi(n) & d' < 0 \end{cases}.$$

A.3 Schnelle Exponentiation

Gegeben sei eine Zahl $x \in \mathbb{Z}_n$ sowie ein Exponent $e \in \mathbb{N}$ mit der binären Darstellung

$$e = \sum_{i=0}^{k} e_i \cdot 2^i \quad \text{mit } e_i \in \{0,1\}.$$

Nun soll $x^e \mod n$ berechnet werden. Es gilt

$$x^e = x^{\sum_{i=0}^{k} e_i \cdot 2^i} = \prod_{i=0}^{k} (x^{2^i})^{e_i} = \prod_{\substack{0 \leq i \leq k \\ e_i = 1}} x^{2^i},$$

daraus gewinnt man die folgende Idee:

Algorithmus A.6 (Wiederholtes Quadrieren und Multiplizieren)

(1) Berechne die sukzessiven Quadrate $x^{2^i} \mod n$ für $i \in \{0, \ldots, k\}$.
Beachte: $x^{2^{i+1}}$ kann aus x^{2^i} mittels einer Quadrierung berechnet werden.

(2) Bestimme $x^e \mod n$ als modulares Produkt derjenigen $x^{2^i} \mod n$, für die $e_i = 1$ ist.

Der Algorithmus führt also genau k Quadrierungen modulo n und höchstens k modulare Multiplikationen durch. Eine primitive Berechnung hätte e Multiplikationen modulo n benötigt. Es ist $k = \lfloor \log_2 e \rfloor + 1 \ll e$, der hier dargestellte Algorithmus ist also schneller.

B Anhang

B.1 Schlusswort

Abschließend möchte ich mich zum einen bei Frau Prof. Dr. Ulrike Baumann und zum anderen bei Matthias Lange für ihre Unterstützung ganz herzlich bedanken!
Das Kapitel A.1 *Primzahlerzeugung* wurde mir von Wieland Marth zur Verfügung gestellt, auch dafür vielen Dank!

B.2 Literaturangaben

Meine verwendeten Bücher sind nachfolgend aufgelistet.

- *Einführung in die Kryptographie* von J. Buchmann (Springer)
- *Cryptography* von D.R. Stinson (Chapman & Hall / CRC Press)
- *Handbook of Applied Cryptography* von A. Menezes, P. van Oorschot und S. Vanstone (CRC Press)
- *Kryptologie* von F.L. Bauer (Springer)
- *Kryptologie* von A. Beutelspacher (Vieweg)