Die Mathematik von RSA

vom ggT zu gpg

Lars Fischer[1]

30.05.2012

[1] lars.fischer (bei) gmx-topmail.de

Inhaltsverzeichnis

Das Worksheet mit den Beispielen

Ich habe das Worksheet auf `http://www.sagenb.org/home/pub/4780` gepublished. Wer dort ein Login anlegt, kann so eine Kopie bearbeiten.
Das gesamte Worksheet ist zusätzlich als Text-Datei in dem PDF eingebettet.

- Im Acrobat-Reader lässt es sich unter dem Büroklammer-Symbol in der linken Leiste herunterladen.
- Okular zeigt es im File-Menu als Embeded Files an.
- Unter Linux kann man die Text-Datei auch mit
  pdftk RSAMath.pdf unpack_files
  aus dem PDF herauslösen.

Anschließend lässt sich die Text-Datei mit der Upload-Funktion des SAGE-Notebooks hochladen.

Schlüsselerzeugung bei Bob

Alice möchte Bob eine verschlüsselte Nachricht zukommen lassen.
Bob wählt:

- zufällig zwei große Primzahlen p, q, deren Produkt $n := pq$, sowie $\varphi := (p-1)(q-1)$
- zufällig eine natürliche Zahl e mit $1 < e < \varphi$, die teilerfremd zu φ ist
- Daraus berechnet Bob eine natürliche Zahl d mit $1 < d < \varphi$, die mit einer beliebigen ganzen Zahl t diese Bedingung erfüllt:

$$de = \varphi t + 1. \tag{1}$$

Der öffentliche Schlüssel ist das Paar (n, e), der private Schlüssel ist die Zahl d.

Verschlüsselung bei Alice

- „Nachricht“ m ist eine natürliche Zahl mit $0 \leq m < n$
- mit Bobs öffentlichen Schlüssel (n, e) berechnet Alice die chiffrierte Nachricht $c = m^e \mod n$

Entschlüsselung bei Bob

Bob berechnet mit seinem privaten Schlüssel d die Zahl

$$m' := c^d = (m^e)^d = m^{ed} = m^{\varphi t+1} = (m^\varphi)^t m \mod n.$$

Hier geschieht nun ein kleines Wunder

„Aufgrund der Mathematik“, gilt

$$m^\varphi \equiv 1 \pmod{n}, \tag{2}$$

deswegen ist $m' \equiv 1^t m \equiv m \pmod{n}$.

Die Schlüsselerzeugung und die Verschlüsselung sind natürlich so gewählt, dass die Entschlüsselung funktioniert.

Das Rechnen mit Resten

- die Zeichen $\equiv$ und $\pmod{n}$: keine Gleichung zwischen Zahlen
- RSA rechnet mit Resten bzgl. der Division durch n
- die konkrete Zahl interessiert nicht, nur der Rest nach Division durch n
- deswegen wachsen die Potenzen $(m^e)^d$ nicht ins Unermessliche, sie werden immer wieder in den Bereich von $0, \ldots, n-1$ „hineingeworfen“

Erst einmal viele Beispiele

1. Wie spät ist es jetzt, und wie spät ist es in 7 Stunden?
2. **Beispiel:** Rechnen mit Resten in SAGE ▸ Link
3. **Beispiel:** Verknüpfungstabellen der Addition und Multiplikation mod n ▸ Link
4. Im vorigen Beispiel mit $n = 2$: Addition ist XOR, Multiplikation ist AND
5. **Beispiel:** die Potenzen bleiben klein ▸ Link
6. **Versuch einer Visualisierung:** ▸ Link

Allgemeines zu den Rechenregeln

Für das Rechnen mit den Resten gelten die gleichen Rechenregel (Buchstabenrechnen), wie für die Ganzen Zahlen. Ihr könnt Euch das so vorstellen: zuerst wird die Rechnung in den Ganzen Zahlen ausgeführt und erst ganz am Schluss die Reste berechnet.
Es macht aber für das Endergebnis keinen Unterschied, wenn wir schon während der Rechnung zu Resten übergehen.

Beispiel: Rechenreihenfolge ▸ Link

Buchstabenrechnen

Es seien a, b, c jeweils Reste mod n

- $ab \equiv ba \pmod{n}$ (d.h. die Verknüpfungstabelle der Multiplikation ist symmetrisch zur Hauptdiagonalen)
- $a(bc) \equiv (ab)c \pmod{n}$

Beispiel

Es sei $\alpha = nx + a$, und $\beta = ny + b$. Dabei seien a und b die Reste bei Division durch n. Dann ist

$$\begin{aligned}\alpha\beta &= (nx + a)(ny + b) = nxny + nxb + nya + ab \\ &= n(nxy + xb + ya) + ab = nz + ab.\end{aligned}$$

Man sieht $\alpha\beta \equiv ab \pmod{n}$, man hätte direkt mit den Resten rechnen können.

Auf diese Weise kann man die Regeln des Buchstabenrechnens leicht nachprüfen.

Ähnliches gilt für die Addition von Resten.

Uns interessiert wirklich nur der Rest nicht die konkrete Zahl

Angenommen in dem vorigen Beispiel hätten wir andere Zahlen α', β' mit den gleichen Resten gehabt, dann wäre das Ergebnis der Multiplikation immer noch ab:

$$\alpha'\beta' = (nx' + a)(ny' + b) \equiv ab \pmod{n}$$

Wir werfen hier *alle* Zahlen, die den gleichen Rest a bei Division durch n haben in einen großen Topf. (Erinnert Euch an die Zahlen, die auf derselben Kante des Zylinders lagen.)

Die Restklassen

Die Mathematiker haben für diesen Topf einen ganz bestimmten Namen:

- Die Restklasse $a \pmod n$ ist die Menge aller Zahlen, die den Rest a bei Division durch n lassen, in Symbolen:

$$\{x | x \equiv a \pmod n\} = a + n\mathbb{Z}.$$

- Weil es die Reste $0, \ldots, n-1$ gibt, haben wir n verschiedene Restklassen

$$0 + n\mathbb{Z}, 1 + n\mathbb{Z}, \ldots, (n-1) + n\mathbb{Z}.$$

Da $n \equiv 0 \pmod n$ ist können wir stattdessen auch $1 + n\mathbb{Z}, \ldots, n + n\mathbb{Z}$ schreiben.

Der größte gemeinsame Teiler

Ein gemeinsamer Teiler zweier Zahlen $a, b > 0$ ist eine Zahl d, die sowohl a als auch b teilt.
Der größte gemeinsame Teiler ist derjenige gemeinsame Teiler, der am größten ist:

$$\mathrm{ggT}(a, b) := \max\{d : d|a \text{ und } d|b\}$$

Eigenschaften des ggT

- Es gibt ganze Zahlen x, y mit $ax + by = \mathrm{ggT}(a, b)$.
- Die Menge aller ganzzahligen Linearkombinationen von a und b ist die Menge aller ganzzahligen Vielfachen von $\mathrm{ggT}(a, b)$:

$$a\mathbb{Z} + b\mathbb{Z} = \mathrm{ggT}(a, b)\mathbb{Z}.$$

Beispiel: ggT ▸ Link

Der euklidische Algorithmus

Der ggT wird mit dem euklidischen Algorithmus (ca. 300 v. Chr.) berechnet. Dieser ist der älteste bekannte Algorithmus. Grundlage für die Berechnung sind diese beiden Eigenschaften des ggT:

- $b = 0 : \mathrm{ggT}(a, b) = |a|$
- $b \neq 0 : \mathrm{ggT}(a, b) = \mathrm{ggT}(|b|, a \mod |b|)$

Der Algorithmus ist sehr effizient, die Anzahl der Ziffern der kleineren der beiden Zahlen bestimmt die Anzahl der benötigten Schritte.

Beispiel: Der Algorithmus in SAGE ▸ Link

Der erweiterte euklidische Algorithmus

Wir hatten oben gesagt, dass $a\mathbb{Z} + b\mathbb{Z} = \mathrm{ggT}(a, b)\mathbb{Z}$ gilt. D.h. es gibt zwei Zahlen x, y mit

$$ax + by = \mathrm{ggT}(a, b).$$

Indem wir im euklidischen Algorithmus genau Buch darüber führen, welche Berechnungen ausgeführt wurden, erhalten wir die Zahlen x, y. Das ist der erweiterte euklidischer Algorithmus.

Beispiel: Der erweiterte Algorithmus in SAGE ▸ Link

Eulersche φ Funktion

Wir betrachten wieder die Reste *mod n*. Ist a teilerfremd zu n, dann ist $\mathrm{ggT}(a, n) = 1$ und wir können mit dem erweiterten euklidischen Algorithmus Zahlen a' und t berechnen, so dass gilt: $aa' + nt = 1$.
Das bedeutet aber: $aa' \equiv 1 \pmod{n}$. Mit einem solchen a' können wir eine Gleichung $ax \equiv b \pmod{n}$ lösen:

$$aa'x \equiv a'b \pmod{n} \Leftrightarrow 1x \equiv a'b \pmod{n}.$$

Wir nennen ein solches a' das Inverse zu a mod n und bezeichnen es als a^{-1}.

Umgekehrt bedeutet $\mathrm{ggT}(a, n) > 1$: egal, was wir auch versuchen, es wird uns niemals gelingen ein Inverses a^{-1} (und ein t) zu finden, so dass obige Gleichung erfüllt ist.

Eulersche φ Funktion

Für ein beliebiges n sind nur einige Restklassen invertierbar. Die Eulersche φ Funktion gibt die Anzahl der invertierbaren Elemente an:

$$\varphi(n) := \#\{a + n\mathbb{Z} : \mathrm{ggT}(a, n) = 1\}$$

Beispiel in SAGE ▸ Link

Eulersche φ Funktion

Für eine Primzahl p, sind alle Reste $1, \ldots, p-1$ teilerfremd zu p, für eine Primzahl p ist deswegen $\varphi(p) = p - 1$. Damit sind auch alle Restklassen $\neq 0$ invertierbar.

Allgemein: Sei $n = p_1^{k_1} \cdots p_n^{k_n}$ die Primfaktorzerlegung von n, dann berechnet sich

$$\varphi(n) = (p_1^{k_1} - p_1^{k_1-1}) \cdots (p_n^{k_n} - p_n^{k_n-1}).$$

Im Fall $n = pq$ ist $\varphi(n) = (p-1)(q-1)$.

Die kleine Satz von Fermat

Der kleiner Satz von Fermat

Es sei p eine Primzahl: Für alle ganzen Zahlen m gilt

$$m^p \equiv m \pmod{p}.$$

Ist m kein Vielfaches von p, so gilt weiter: $m^{p-1} \equiv 1 \pmod{p}$.

Beispiel: Satz von Fermat ▸ Link

Beweis, dass die RSA Entschlüsselung funktioniert

Wir beweisen nun die RSA Verschlüsselung mit dem vorigen Satz.
Die Zahl n teilt nur dann $m^{ed} - m$, wenn p *und* q beide $m^{ed} - m$ teilen. Wir betrachten deswegen die Gleichung mod p und mod q anstatt mod n.

Fallunterscheidung

1. $\mathrm{ggT}(m, p) > 1$: auf beiden Seiten der RSA-Entschlüsselungsformel (2) stehen Vielfache von p, beide Seiten sind $\equiv 0 \pmod{p}$ (und damit mod p gleich).
2. $\mathrm{ggT}(m, p) = 1$: so ist nach vorigem Satz $m^{ed} \equiv m^{\varphi t+1} \equiv m^{(p-1)(q-1)t} m \equiv m \pmod{p}$.

Auf die gleiche Weise folgt die Äquivalenz mod q.

Beweis, dass die RSA Entschlüsselung funktioniert

Wir haben gezeigt:

$$m^{ed} - m = t_1 p$$
$$m^{ed} - m = t_2 q$$

D.h. $m^{ed} - m$ ist ein Vielfaches der Primzahlen p und q, und damit ein Vielfaches von n:
Also ist $m^{ed} - m = t_3 n$, das bedeutet $m^{ed} \equiv m \pmod{n}$. □

RSA zum Zweiten.

Beispiel: RSA ▸ Link

Zum Abschluss noch einmal ein Blick auf die RSA-Schlüsselerzeugung und RSA-Entschlüsselung ...

Schlüsselerzeugung bei Bob

Alice möchte Bob eine verschlüsselte Nachricht zukommen lassen.
Bob wählt:

- zufällig zwei große Primzahlen p, q, deren Produkt $n := pq$, sowie $\varphi := (p-1)(q-1)$
- zufällig eine natürliche Zahl e mit $1 < e < \varphi$, die teilerfremd zu φ ist
- Daraus berechnet Bob eine natürliche Zahl d mit $1 < d < \varphi$, die mit einer beliebigen ganzen Zahl t diese Bedingung erfüllt:

$$de = \varphi t + 1. \tag{1}$$

Der öffentliche Schlüssel ist das Paar (n, e), der private Schlüssel ist die Zahl d.

Entschlüsselung bei Bob

Bob berechnet mit seinem privaten Schlüssel d die Zahl

$$m' := c^d = (m^e)^d = m^{ed} = m^{\varphi t+1} = (m^\varphi)^t m \mod n.$$

Hier geschieht nun ein kleines Wunder

„Aufgrund der Mathematik“, gilt

$$m^\varphi \equiv 1 \pmod{n}, \tag{2}$$

deswegen ist $m' \equiv 1^t m \equiv m \pmod{n}$.

Die Schlüsselerzeugung und die Verschlüsselung sind natürlich so gewählt, dass die Entschlüsselung funktioniert.

Bemerkungen

Was ich alles nicht erwähnt habe

- Alle Rechenoperationen passen nicht in die Hardware (viele 100 Bits vs. 64 Bit CPUs)
- Primzahltests
- Signaturen
- Hashes
- Andere Public Key Verfahren, als Alternativen, wenn irgendwann große Zahlen faktorisiert werden können (Quantencomputing oder verbesserte Algorithmen)
- Symmetrische Verfahren

Offiziell

- 1976: Whitfield Diffie und Martin Hellman haben die Idee asymmetrischer Chiffrierverfahren
- Public-Key Kryptografie ist geboren
- Diffie / Hellman hatten nur die Idee aber noch kein praktikables Verfahren.
- 1978: Rivest, Shamir und Adleman veröffentlichen das erstes praktikables Public Key Kryptografie Verfahren: RSA

Inoffiziell

- 1970: James Ellis vom britischen Government Communication Headquarters (G.C.H.Q.): ebenfalls die Idee der Public Key Kryptografie
- auch bei ihm fehlt ein praktikables Verfahren
- 1973: Clifford Cocks, ein Mathestudent, der gerade beim G.C.H.Q. angefangen hat, findet ein Verfahren, ähnlich RSA
- keine Veröffentlichung wegen Geheimhaltung
- 1997 „geben die Briten zu“, dass sie die Ersten wahren

Weiterführende Links bzw. Literatur

- Alfred J. Menezes, Paul C. van Oorschot und Scott A. Vanstone: „Handbook of Applied Cryptography“, http://cacr.uwaterloo.ca/hac/
- Simon Singh: „Geheime Botschaften“
- Johannes Buchmann: „Einführung in die Kryptographie“
- Niven, Zuckerman und Montgomery: „An Introduction to the Theory of Numbers“
- RSA bei Wikipedia, http://de.wikipedia.org/wiki/RSA-Kryptosystem
- Esslinger et al.: „Das CryptTool-Skript“, http://www.cryptool.org/images/ctp/documents/CrypT

Ende

Fragen?

Ende

Vielen Dank für Eure Aufmerksamkeit

Mathematische Anhängsel

Hier folgen jetzt noch Erläuterungen zum Kleinen Satz von Fermat.
Dieser Abschnitt ist nur noch für Mathematik-Interessierte genießbar.

Beweis vom kleinen Satz von Fermat

Eine kombinatorische Beweisvarianten findet Ihr unter
http://en.wikipedia.org/wiki/Proofs_of_Fermat's_little

Hier aber ein kompakter Beweis:
Für eine Primzahl p und eine beliebige ganze Zahl m wollen wir zeigen

$$m^p \equiv m \pmod{p}.$$

Falls p ein Teiler von m ist, dann ist $m \equiv 0 \pmod{p}$ und die Gleichung besagt $0^p \equiv 0 \pmod{p}$, sie ist in diesem Fall bereits wahr.
Wir betrachten nun den Fall, dass m und p teilerfremd sind:
Dann gibt es ein $m^{-1} \pmod{p}$ (d.h. wir können durch m teilen, bzw. mit m^{-1} multiplizieren und haben die Gleichung

$$m^{p-1} \equiv 1 \pmod{p}.$$

Beweis vom kleinen Satz von Fermat

Um dies zu zeigen betrachten wir das Produkt aller Restklassen:

$$M := 1 \cdot \ldots \cdot (p-1) \tag{3}$$

Und das Produkt M', mit

$$\begin{aligned} M' := m^{p-1} M &= m^{p-1}(1 \cdot \ldots \cdot (p-1)) \\ &= (m \cdot 1) \cdot \ldots \cdot (m \cdot (p-1)) \end{aligned} \tag{4}$$

Die letzte Zeile ist nur eine Umordnung der Gleichung (3): Die Menge, der mit m multiplizierten Restklassen, stimmt mit der Menge der Restklassen überein, nur die Reihenfolge verändert sich.
Z.B. sind in den Verknüpfungstabellen der Multiplikation für Primzahl-Moduln die späteren Zeilen nur eine Umordnung der Zeile für die Restklasse 1.

Beweis vom kleinen Satz von Fermat

- Oben und unten steht eine Menge von $p-1$ Elementen.
- Wäre das Produkt in (4) keine bloße Umordnung, würde insbesondere unten *kein vollständiges Restesystem* stehen, es würde ein Rest fehlen. Es müsste also in Gleichung (4) Elemente mx, my geben, mit $x \not\equiv y \pmod{p}$ aber $mx \equiv my \pmod{p}$: der Rest mx kommt unten doppelt vor.
- Dann wäre

$$\begin{aligned} mx \equiv my \pmod{p} &\Leftrightarrow mx - my = tp \\ &\Leftrightarrow m(x-y) = tp \end{aligned}$$

- Nach Voraussetzung sind m und p teilerfremd, deswegen muss p bereits ein Teiler von $x-y$ sein. D. h. es ist bereits $x \equiv y \pmod{p}$ im Widerspruch zu unserer Annahme.

Beweis vom kleinen Satz von Fermat

Wir haben gezeigt, dass

$$\begin{aligned} 1 \cdot \ldots \cdot (p-1) &\equiv (m \cdot 1) \cdot \ldots \cdot (m \cdot (p-1)) \\ &\equiv m^{p-1} (1 \cdot \ldots \cdot (p-1)) \pmod{p}. \end{aligned}$$

In dem Produkt $1 \cdot \ldots \cdot (p-1)$ sind alle Zahlen teilerfremd zu p, wir können beide Seite durch jede der Zahlen dividieren und haben dann

$$1 \equiv m^{p-1} \pmod{p}$$

gezeigt für alle m, die teilerfremd zu p sind .

Ende

Vielen Dank für Eure Aufmerksamkeit