Kryptographie II
SS 2007

G. Leander und R. Avanzi
T. Brecher, M. Goldack, O. Grieb, S. Hoerder,
T. Kornau, O. Paustjan, S. Spitz, C. Wachsmann

7. April 2008

Inhaltsverzeichnis

Kapitel 1

RSA

Ein gut bekanntes und wahrscheinlich das am weitesten verbreitete asymmetrisches Verschlüsselungsverfahren ist RSA, welches auf die Anfangsbuchstaben seiner Erfinder Ronald L. **R**ivest, Adi **S**hamir und Leonard **A**dleman zurückgeht. Im Folgenden wird das Verfahren kurz vorgestellt.

1.1 Schlüsselerzeugung

Wähle $p, q \in \mathbb{N}$ prim und berechne den Modulus

$$n = p \cdot q$$

In der Praxis beträgt die Bitlänge:

$$\log_2 p \approx \log_2 q \approx 512$$

Wähle anschließend den öffentlichen Schlüssel

$$e \in \{1, \ldots, n-1\} \qquad \text{mit} \qquad \text{ggT}(e, (p-1)(q-1)) = 1$$

und berechne den geheimen Schlüssel

$$d \equiv e^{-1} \bmod (p-1)(q-1) \qquad \text{d.h.} \qquad ed \equiv 1 \bmod (p-1)(q-1)$$

In der Praxis wird der öffentliche Schlüssel e klein gewählt, zum Beispiel $e = 17$, um effizienter verschlüsseln zu können.

1.2 Verschlüsselung

Die zu verschlüsselnde Nachricht sei:

$$m \in \mathbb{Z}_n$$

Berechne den Chiffretext

$$c = m^e \bmod n$$

Diese Berechnung, und auch die Entschlüsselung, erfolgt in der Regel mit einer effizienten Variante des Square-And-Multiply-Algorithmus

1.3 Entschlüsselung

Zu einem gegebenen Chiffretext $c \in \mathbb{Z}_n$ berechne

$$m' = c^d \bmod n$$

Für den Beweis, dass die Entschlüsselung tatsächlich die vorher verschlüsselte Nachricht liefert, benötigen wir folgenden Satz, der aus einer Einführungsvorlesung bekannt sein sollte.

1.1 Theorem. *Kleiner Fermat*
Sei $a, n \in \mathbb{N}$ dann gilt:

$$a^{\varphi(n)} \equiv 1 \bmod n \qquad \textit{für} \qquad \mathrm{ggT}(a, n) = 1,$$

wobei

$$\varphi(n) = |\{1 \leq x < n \mid \mathrm{ggT}(x, n) = 1\}|$$

die Eulersche φ-Funktion bezeichnet. Insbesondere gilt für $n = pq$ mit p, q prim:

$$a^{(p-1)(q-1)} \equiv 1 \bmod n$$

Ausserdem, auch das hoffentlich nur eine Auffrischung, benötigen wir den Chinesischen Restsatz („Chinese Remainder Theorem“, CRT).

1.2 Theorem. *Chinesischer Restsatz*
Sei $m, n \in \mathbb{N}$ mit $\mathrm{ggT}(m, n) = 1$, dann gilt:

$$\mathbb{Z}_{mn} \cong \mathbb{Z}_m \times \mathbb{Z}_n,$$

d.h. die beiden Ringe $\mathbb{Z}_{mn}$ und $\mathbb{Z}_m \times \mathbb{Z}_n$ sind isomorph.

Beweis: Die Abbildung $\Phi : \mathbb{Z}_{mn} \to \mathbb{Z}_m \times \mathbb{Z}_n$

$$\Phi(x) = (x \bmod m, x \bmod n)$$

ist sicher ein Ringisomorphismus. Es bleibt noch zu zeigen, dass Φ bijektiv ist. Da beide Ringe endlich und gleichmächtig sind, reicht zu zeigen, dass Φ injektiv oder surjektiv ist. Man zeigt, dass Φ surjektiv ist, indem man zu jedem $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_m$ ein Urbild $x \in \mathbb{Z}_{mn}$ konstruiert, also

$$\Phi(x) = (a, b)$$

Wir müssen also ein Element $x \in \mathbb{Z}_{mn}$ finden, so dass gleichzeitig

$$x \equiv a \bmod n \tag{1.1}$$
$$x \equiv b \bmod m \tag{1.2}$$

gelten. Dazu sucht man zuerst ein Element $e_n \in \mathbb{Z}_{mn}$ mit

$$e_n \equiv \begin{cases} 0 \bmod m \\ 1 \bmod n \end{cases}$$

und ein Element $e_m \in \mathbb{Z}_{mn}$ mit

$$e_m \equiv \begin{cases} 0 \bmod n \\ 1 \bmod m \end{cases}$$

Sei nun $e'_n = m$. Dann gelten

$$e'_n \equiv 0 \bmod m$$

und

$$e'_n \neq 0 \bmod n \qquad \text{da} \qquad \text{ggT}(e'_n, n) = 1$$

gilt. Setze:

$$e_n \equiv e'_n(e'^{-1}_n \bmod n) \equiv m(m^{-1} \bmod n)$$

Dann gilt:

$$e_n \equiv e'_n(e'^{-1}_n \bmod n) \equiv 1 \bmod n$$

Analog setze:

$$e_m \equiv n(n^{-1} \bmod m)$$

Dann erfüllt

$$x = a \cdot e_n + b \cdot e_m$$

die Gleichungen (1.1) und (1.2), denn

$$\begin{aligned} x &\equiv a(e_n \bmod n) + b(e_m \bmod n) \\ &\equiv a \bmod n \end{aligned}$$

und analog

$$x \equiv b \bmod m$$

Damit ist gezeigt, dass jedes Element $(a, b) \in \mathbb{Z}_m \times \mathbb{Z}_n$ ein Urbild unter Φ hat. Das heisst, Φ ist surjektiv und damit bijektiv. □

Der Beweis liefert auch eine Methode um $\Phi^{-1}(a, b)$ praktisch zu berechnen.

Mit den Theoremen 1.1 und 1.2 können wir nun folgenden Satz zeigen:

1.3 Theorem. *Sei $n = pq$ und $ed \equiv 1 \bmod (p-1)(q-1)$. Dann gilt für alle $x \in \mathbb{Z}_n$*

$$(x^e)^d \equiv x \bmod n$$

Beweis: Aus dem Satz von Fermat folgt, das für jede Primzahl p und alle natürliche Zahlen a, t

$$a^{1+t(p-1)} \equiv a \bmod p$$

gilt. Mit Hilfe des CRT berechnen wir daher

$$\begin{aligned}
(x^e)^d &\equiv \Phi^{-1}\left(\Phi(x^{ed})\right) \\
&\equiv \Phi^{-1}\left(x^{ed} \bmod p, x^{ed} \bmod q\right) \\
&\equiv \Phi^{-1}\left(x^{ed} \bmod p, x^{ed} \bmod q\right) \\
&\equiv \Phi^{-1}\left(x^{1+t_1(p-1)(q-1)} \bmod p, x^{1+t_2(p-1)(q-1)} \bmod q\right) \\
&\equiv \Phi^{-1}\left(x \bmod p, x \bmod q\right) \\
&\equiv x
\end{aligned}$$

□

1.4 Einiges über Ordnungen von Elementen

Für eine endliche abelsche Gruppe G und $x \in G$ definiert man mittels

$$\mathrm{ord}(x) = \min\{r \in \mathbb{N}\setminus\{0\} \mid x^r = 1\}$$

die *Ordnung* von x. Im Falle von $x, n \in \mathbb{Z}$ mit $\mathrm{ggT}(x, n) = 1$ bezeichne

$$\mathrm{ord}_n(x) = \min\{r \in \mathbb{N}\setminus\{0\} \mid x^r \equiv 1 \bmod n\},$$

die Ordnung von x in $\mathbb{Z}_n$.

1.4 Lemma. *Sei G eine endliche abelsche Gruppe und $x \in G$ dann gilt:*

$$\mathrm{ord}(x) \mid |G| \tag{1.3}$$

$$x^k = 1 \Leftrightarrow \mathrm{ord}(x) | k \tag{1.4}$$

$$\mathrm{ord}(x^n) = \frac{\mathrm{ord}(x)}{\mathrm{ggT}(\mathrm{ord}(x), n)} \tag{1.5}$$

Beweis Die erste Aussage folgt aus dem Satz von Lagrange, der besagt, dass die Ordnung jeder Untergruppe* die Ordnung der Gruppe teilt mit der Beobachtung, dass

$$|\langle x\rangle| = \mathrm{ord}(x).$$

√

*$\langle x\rangle$ bezeichnet die von $x \in G$ erzeugte Untergruppe von G

Für die zweite Aussage sind zwei Implikationen zu beweisen. Zunächst wird die Richtung $\Leftarrow$ gezeigt. Sei k mit $\mathrm{ord}(x)|k$ gegeben. Das heißt:

$$k = s \cdot \mathrm{ord}(x) \qquad \text{mit} \qquad s \in \mathbb{N}$$

Dann gilt

$$x^k = x^{s \cdot \mathrm{ord}(x)} = (x^{\mathrm{ord}(x)})^s = 1^s = 1$$

Jetzt wird die Richtung $\Rightarrow$ gezeigt. Sei $k \in \mathbb{N}$ mit $x^k = 1$ gegeben, und sei

$$k = s \cdot \mathrm{ord}(x) + r \qquad \text{mit} \qquad 0 \leqslant r < \mathrm{ord}(x)$$

Wir wollen zeigen dass $r = 0$ folgt. Es gilt:

$$1 = x^k = x^{s \cdot \mathrm{ord}(x) + r} = (x^{\mathrm{ord}(x)})^s \cdot x^r = x^r$$

Da $r < k$ und $\mathrm{ord}(x)$ die kleinste positive Zahl ist, so dass $x^{\mathrm{ord}(x)} = 1$ gilt folgt $r = 0$ und damit gilt $\mathrm{ord}(x)|k$. $\surd$

Für die dritte Behauptung setze $e = \mathrm{ord}(x)$. Es gilt:

$$(x^n)^{\frac{e}{\mathrm{ggT}(e,n)}} = (x^e)^{\frac{n}{\mathrm{ggT}(e,n)}} = 1^{\frac{n}{\mathrm{ggT}(e,n)}} = 1$$

Nach 1.4 folgt:

$$\mathrm{ord}(x^n) | \frac{e}{\mathrm{ggT}(e,n)} \tag{1.6}$$

Sei nun $k \in \mathbb{N}$, so dass $(x^n)^k = 1$, dann gilt

$$x^{nk} = 1$$

und mit 1.4 folgt daraus: $e|nk$. Damit gilt

$$\frac{e}{\mathrm{ggT}(e,n)} | k \Rightarrow \frac{e}{\mathrm{ggT}(e,n)} | \mathrm{ord}(x^n)$$

Mit 1.6 folgt:

$$\frac{e}{\mathrm{ggT}(e,n)} = \mathrm{ord}(x^n)$$

$\square$

1.5 Lemma. *Sei $n = pq$ mit p, q prim und $p, q \neq 2$. Dann gilt für $x \in \mathbb{Z}_n^*$*

$$\mathrm{ord}_n(x) = \mathrm{kgV}(\mathrm{ord}_p(x), \mathrm{ord}_q(x))$$

Beweis: Nach dem Chinesischen Restsatz gibt es einen Ringisomorphismus

$$\Phi : \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q$$

Dies induziert einen Gruppenisomorphismus

$$\Phi : \mathbb{Z}_n^* \to \mathbb{Z}_p^* \times \mathbb{Z}_q^*$$

Es gilt:

$$\begin{aligned} x^{\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))} &= \Phi^{-1}(\Phi(x^{\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))})) \\ &= \Phi^{-1}(x^{\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))} \bmod p, x^{\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))} \bmod q) \end{aligned}$$

Aus der Definition des kgV folgt unmittelbar, dass $p-1|\,\mathrm{kgV}(p-1,q-1)$ sowie $q-1|\,\mathrm{kgV}(p-1,q-1)$ und mit dem Satz von Fermat ergibt sich

$$\begin{aligned} x^{\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))} &= \Phi^{-1}(1,1) \\ &= 1 \end{aligned}$$

Daraus folgt

$$\mathrm{ord}_n(x)|\,\mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x)).$$

Sei nun $k \in \mathbb{N}$, so dass $x^k \equiv 1 \bmod n$.

$$\begin{aligned} &\Rightarrow \Phi^{-1}(x^k \bmod p, x^k \bmod q) = 1 \\ &\Rightarrow x^k \bmod p \equiv 1 \wedge x^k \bmod q \equiv 1 \\ &\Rightarrow \mathrm{ord}_p(x)|k \wedge \mathrm{ord}_q(x)|k \\ &\Rightarrow \mathrm{kgV}(\mathrm{ord}_p(x),\mathrm{ord}_q(x))|k \end{aligned}$$

□

Eine unmittelbare Folgerung aus Lemma 1.5 ist, dass für $n = pq$ mit $p, q \neq 2$ und p, q prim die Gruppe $\mathbb{Z}_n^*$ nicht zyklisch ist, dass heißt es gibt keinen Generator. Ein Generator müsste die Ordnung $\phi(n) = (p-1)(q-1)$ haben. Für $x \in \mathbb{Z}_n^*$ gilt:

$$\mathrm{ord}_p(x)|(p-1) \wedge \mathrm{ord}_q(x)|(q-1)$$

Nach obigem Satz gilt

$$\mathrm{ord}_n(x)|\,\mathrm{kgV}(p-1,q-1)$$

Da $p-1, q-1 \equiv 0 \bmod 2$ folgt:

$$\mathrm{kgV}(p-1,q-1) \leqslant \frac{(p-1)(q-1)}{2}$$

und damit

$$\mathrm{ord}_n(x) < (p-1)(q-1)$$

1.5 Sicherheit von RSA

Es ist klar, dass ein Angreifer, der den Modul n faktorisieren kann, auch RSA brechen kann, da er dann den geheimen Exponenten d berechnen kann. Die Frage, ob ein Angreifer, der RSA brechen kann, auch n faktorisieren kann, ist jedoch ein offenes Problem. Wir werden in diesem Abschnitt aber zwei Methoden sehen, die mit gegebenem n, e, d und $ed \equiv 1 \bmod \varphi(n)$ den Modulus n faktorisieren können. Das heißt, dass das Berechnen des geheimen Exponenten äquivalent zum Faktorisieren von n ist. Es ist aber durchaus denkbar, dass man RSA brechen kann, ohne den geheimen Exponenten zu kennen - auch wenn heute wahrscheinlich Niemand einen solchen Algorithmus kennt.

1.5.1 Erste Methode: Ein analytischer Angriff

Seien im Folgenden $n = pq$ und e, d mit

$$ed = 1 \bmod \varphi(n) \tag{1.7}$$

und

$$n = pq, \qquad p, q \in \mathbb{P}$$

gegeben.

Zuerst wird folgendes Lemma benötigt:

1.6 Lemma. *Für $n = pq$ mit $p \neq q$ und $p, q \in \mathbb{P}$ gilt:*

$$p + q > 2\sqrt{n}$$

Beweis: Durch Umformen erhält man:

$$\begin{aligned} 2\sqrt{n} &< p + q \\ 4n &< (p+q)^2 \\ 4pq &< p^2 + 2pq + q^2 \\ 2pq &< p^2 + q^2 \end{aligned}$$

Nun kann man o.B.d.A. annehmen, dass $p > q$ gilt und dass man somit p durch $q + r$ mit $r \geqslant 1$ substituieren kann. Damit erhält man durch weiteres Umformen:

$$\begin{aligned} 2(q+r)q &< (q+r)^2 + q^2 \\ 2q^2 + 2qr &< 2q^2 + 2qr + r^2 \\ 0 &< r^2 \end{aligned}$$

□

Nun folgt aus Gleichung 1.7:

$$\begin{aligned} ed &= 1 + k(p-1)(q-1) \\ &= 1 + k(pq - (p+q) + 1) \\ ed - 1 &= k(n - (p+q) + 1) \end{aligned}$$

Um n zu faktorisieren, wird eine Abschätzung für den Term $\sigma(k) = p + q$ benötigt. Diese erhält man folgendermaßen:

$$\begin{aligned} ed &= 1 + k(n - (p+q) + 1) \\ \frac{ed-1}{k} &= n - \underbrace{(p+q)}_{\sigma(k)} + 1 \\ \sigma(k) &> -\frac{ed-1}{k} + n + 1 \end{aligned}$$

Nun fehlt nur noch k. Dies kann man mit Hilfe des Lemmas approximieren und erhält somit:

$$\begin{aligned} ed - 1 &< k(n - 2\sqrt{n} + 1) \\ k &> \frac{ed-1}{n - 2\sqrt{n} + 1} \end{aligned}$$

Daraus ergibt sich nun der folgende Algorithmus:

1.7 Algorithmus. ***Einfache RSA Faktorisierung 2***

- *Setze* $k \leftarrow \lceil \frac{ed-1}{n-2\sqrt{n}+1} \rceil$.
- *Teste, ob* $k \mid ed - 1$ *gilt. Wenn ja:*
  - *Setze* $\sigma(k) \leftarrow p + q = -\frac{ed-1}{k} + n + 1 \in \mathbb{N}$.
  - *Löse die quadratische Gleichung* $x^2 - \sigma(k)x + n = 0$.
  - *Wenn* $x_1, x_2 \notin \{1, n\}$ *gebe* $p = x_1$ *und* $q = x_2$ *aus und stoppe.*
- *Setze* $k \leftarrow k + 1$.
- *Gehe zu* 2.

1.8 Beispiel. *Seien* $n = 697$ *und* $(e, d) = (3, 427)$ *gegeben. Dann erhält man:*

$$k_0 = \left\lceil \frac{3 \cdot 427 - 1}{697 - 2\sqrt{697} + 1} \right\rceil = 2$$

Da $2 \mid ed-1$ *gilt, erhält man:*

$$\begin{aligned}\sigma(k_0) &= -\frac{3\cdot 427-1}{2}+697+1\\ &= -640+697+1\\ &= 58\end{aligned}$$

Daraus erhält man die quadratische Gleichung $x^2-58x+697=0$*, die die Lösungen*

$$x_1, x_2 = \frac{58\pm\sqrt{58^2-4\cdot 697}}{2} = \begin{cases}41\\17\end{cases}$$

hat. Da $x_1, x_2 \notin \{1, n\}$ *sind, bekommt man als Ergebnis* $p=41$ *und* $q=17$*.* $41\cdot 17 = 697 = n$ *zeigt, dass das Ergebnis stimmt.*

1.5.2 Zweite Methode: Die algebraische Methode

Seien im Folgenden $n=pq$ und e, d mit

$$ed \equiv 1 \bmod \varphi(n)$$

gegeben. Seien weiterhin s, u bekannt, so dass

$$ed-1 = 2^s\cdot u \qquad \text{mit u ungerade}$$

gilt.

1.9 Lemma. *Sei* $a\in\mathbb{Z}_n^*$*. Dann gilt:*

$$\mathrm{ord}_n(a^u) = 2^i \qquad \textit{für ein} \qquad i\in\{0,\ldots,s\}$$

Beweis: Es gilt

$$(a^u)^{2^s} = a^{u\cdot 2^s} = a^{ed-1} \equiv 1 \bmod n$$

mit Lemma 1.4 folgt daraus

$$\mathrm{ord}_n(a^u) | 2^s.$$

Da die einzigen Teiler von 2^s Potenzen von 2 sind erhalten wir

$$\mathrm{ord}_n(a^u) = 2^i \qquad \text{für} \qquad i\in\{0,\ldots,s\}.$$

□

1.10 Lemma. *Wenn für* $a\in\mathbb{Z}_n^*$ *gilt*

$$\mathrm{ord}_p(a^u) \neq \mathrm{ord}_q(a^u)$$

dann existiert ein $j\in\{0,\ldots,s-1\}$*, so dass*

$$\mathrm{ggT}((a^u)^{2^j}-1, n) \in \{p, q\}$$

Beweis: Sei $\mathrm{ord}_p(a^u) = 2^\alpha$ und $\mathrm{ord}_q(a^u) = 2^\beta$ mit $\alpha, \beta \in \{0, \ldots, s\}$. Wir nehmen ohne Einschränkung $\alpha < \beta$ an, dann gilt

$$\begin{aligned} (a^u)^{2^\alpha} &\equiv 1 \bmod p \\ (a^u)^{2^\alpha} &\not\equiv 1 \bmod q \end{aligned}$$

also

$$\begin{aligned} (a^u)^{2^\alpha} - 1 &\equiv 0 \bmod p \\ (a^u)^{2^\alpha} - 1 &\not\equiv 0 \bmod q \end{aligned}$$

Daraus folgt

$$\mathrm{ggT}((a^u)^{2^\alpha} - 1, n) = p$$

□

Aus Lemma 1.10 ergibt sich folgender Algorithmus, um n zu faktorisieren:

1.11 Algorithmus. ***Einfache RSA Faktorisierung 1***

- *Wähle* $a \in \mathbb{Z}_n \setminus \{0\}$
- *Teste, ob* $\mathrm{ggT}(a, n) = 1$*: Wenn nicht, gebe* $\mathrm{ggT}(a, n)$ *aus*
- *Berechne* $a^u \bmod n$
- *Für alle* $j \in \{0, \ldots, s-1\}$ *berechne* $\mathrm{ggT}((a^u)^{2^j} - 1, n) = g$
- *Wenn* $g \neq 1$ *und* $g \neq n$*, gebe* g *aus*
- *Wenn der Algorithmus bis hierher keinen Fakor ausgegeben hat, hat man Pech gehabt und muß mit einem neuen Element* a *von vorne beginnen.*

Dieser Algorithmus funktioniert nur für $a \in \mathbb{Z}_n^*$ mit $\mathrm{ord}_p(a^u) \neq \mathrm{ord}_q(a^u)$. Damit dieser Algorithmus erwartete polynomielle Laufzeit hat müssen wir daher sicherstellen, dass der Anteil der Elemente $a \in \mathbb{Z}_n^*$ mit dieser Eigenschaft groß genug ist.

1.12 Lemma. *Die Anzahl der Elemente* $a \in \mathbb{Z}_n^*$*, so dass*

$$\mathrm{ord}_p(a^u) \neq \mathrm{ord}_q(a^u)$$

gilt, ist mindestens

$$\frac{(p-1)(q-1)}{2}$$

Beweis: Sei $g \in \mathbb{Z}_n^*$, so dass $(g \bmod p)$ ein Generator von $\mathbb{Z}_p^*$ und $(g \bmod q)$ ein Generator von $\mathbb{Z}_q^*$. Wir unterscheiden drei Fälle.

1. Fall:

$$\mathrm{ord}_p(g^u) > \mathrm{ord}_q(g^u)$$

Sei $a \in \mathbb{Z}_n^*$ mit

$$\begin{array}{rcll} a & \equiv & g^x \bmod p & \text{mit} \quad x \in \{1,\ldots,p-1\} \quad \text{ungerade und} \\ a & \equiv & g^y \bmod q & \text{mit} \quad y \in \{1,\ldots,q-1\} \quad \text{beliebig} \end{array}$$

Aufgrund des Chinesischen Restsatzes gibt es für jedes Paar (x, y) genau ein Element $a \in \mathbb{Z}_n^*$ mit dieser Eigenschaft. Es gilt

$$\mathrm{ord}_p(a^u) = \mathrm{ord}_p((g^x)^u) = \mathrm{ord}_p((g^u)^x) = \frac{\mathrm{ord}_p(g^u)}{\mathrm{ggT}(\mathrm{ord}_p(g^u), x)} = \mathrm{ord}_p(g^u)$$

da

$$\mathrm{ord}_p(g^u) = 2^j \qquad \text{für} \qquad j \in \{0,\ldots,s\}$$

Weiter gilt:

$$\mathrm{ord}_q(a^u) = \mathrm{ord}_q((g^u)^y) \leqslant \mathrm{ord}_q(g^u)$$

Daraus folgt:

$$\mathrm{ord}_p(a^u) = \mathrm{ord}_p(g^u) > \mathrm{ord}_q(g^u) \geqslant \mathrm{ord}_q((g^u)^y) = \mathrm{ord}_q(a^u)$$

so dass gilt:

$$\mathrm{ord}_p(a^u) \neq \mathrm{ord}_q(a^u)$$

Es gibt also genau $\frac{(p-1)}{2} \cdot (q-1)$ Paare (x, y) mit x ungerade und y beliebig und daher mindestens $\frac{(p-1)(q-1)}{2}$ Elemente a mit der gewünschten Eigenschaft.

2. Fall:

$$\mathrm{ord}_p(g^u) < \mathrm{ord}_q(g^u)$$

Analog zum ersten Fall, wovon sich der geneigte Leser im Rahmen einer kleinen Übungsaufgabe überzeugen mag :-).

3. Fall:

$$\mathrm{ord}_p(g^u) = \mathrm{ord}_q(g^u)$$

Sei $a \in \mathbb{Z}_n^*$ mit

$$\begin{array}{rcll} a & = & g^x \bmod p & \text{mit} \quad x \in \{2,\ldots,p-1\} \quad \text{gerade} \wedge \\ a & = & g^y \bmod q & \text{mit} \quad y \in \{1,\ldots,q-1\} \quad \text{ungerade} \end{array}$$

Also gilt:

$$\mathrm{ord}_q(a^u) = \mathrm{ord}_q((g^y)^u) = \mathrm{ord}_q((g^u)^y) = \frac{\mathrm{ord}_q(g^u)}{\mathrm{ggT}(\mathrm{ord}_q(g^u), y)} = \frac{\mathrm{ord}_q(g^u)}{1} = \mathrm{ord}_q(g^u)$$

und

$$\mathrm{ord}_p(a^u) = \mathrm{ord}_p((g^u)^x) < \mathrm{ord}_p(g^u)$$

Daraus folgt:

$$\mathrm{ord}_p(a^u) < \mathrm{ord}_p(g^u) = \mathrm{ord}_q(g^u) = \mathrm{ord}_q(a^u)$$

Es gibt also genau $\frac{(p-1)}{2} \cdot \frac{(q-1)}{2}$ Paare (x, y) mit x gerade und y ungerade. Analog gilt dies für x ungerade und y gerade. Damit beträgt die Anzahl der Elemente a

$$2 \cdot \frac{(p-1)}{2} \cdot \frac{(q-1)}{2} = \frac{(p-1)(q-1)}{2}$$

□

1.6 Angriff auf RSA mit kleinem öffentlichem Exponenten

Diesen Angriff wollen wir an einem Beispiel demonstrieren: Gegeben seien 3 öffentliche RSA-Schlüssel, wobei der Exponent jeweils 3 ist:

$$(n_1, 3), (n_2, 3), (n_3, 3)$$

A verschlüsselt eine Nachricht

$$m < \min\{n_1, n_2, n_3\}$$

für alle 3 öffentlichen Schlüssel. D.h. A berechnet:

$$c_1 = m^3 \bmod n_1, \quad c_2 = m^3 \bmod n_2, \quad c_3 = m^3 \bmod n_3$$

Ein Angreifer kann aus c_1, c_2, c_3 und den öffentlichen Schlüsseln n_1, n_2, n_3 die Nachricht m berechnen. Dazu berechnet er mit dem CRT ein Element c, so dass

$$c = c_1 \bmod n_1 \tag{1.8}$$
$$c = c_2 \bmod n_2 \tag{1.9}$$
$$c = c_3 \bmod n_3 \tag{1.10}$$

Mit dem CRT gibt es genau ein solches Element

$$c \bmod n_1, n_2, n_3$$

Insbesondere gibt es genau ein c mit

$$0 \leqslant c < n_1 n_2 n_3$$

dass die Gleichungen (1.8), (1.9) und (1.10) erfüllt. Dies ist genau m^3, da

$$m < min\{n_1, n_2, n_3\} \quad \Rightarrow \quad m^3 < n_1 n_2 n_3$$

In den ganzen Zahlen ist aber, im Unterschied zu $\mathbb{Z}_n^*$, das Berechnen von 3. Wurzeln leicht. Damit erhält man

$$m = \sqrt[3]{c}$$

1.7 Angriff auf RSA mit kleinen privaten Exponenten

Michael J. Wiener hat 1989 in [3] einen Angriff demonstriert, der bei RSA für kleine private Schlüssel eine relativ einfache Faktorisierung von n_{RSA} ermöglicht. Um den Angriff erklären zu können, müssen jedoch erst Kettenbrüche eingeführt werden.

1.7.1 Kettenbrüche

Seien q_i für $i \in \{0, \ldots, m\}$ mit $q_m \neq 0$ natürliche Zahlen. Ausdrücke der Form

$$x = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{\ddots + \frac{1}{q_m}}}}}$$

werden als Kettenbruch bezeichnet. In Kurzschreibweise schreiben wir Kettenbrüche als:

$$x = \langle q_0, q_1, \ldots, q_m \rangle$$

Kettenbrüche haben zahlreiche Anwendungen in der Zahlentheorie. Sie bieten insbesondere eine Möglichkeit, reelle Zahlen gut zu approxmimeren. Wir werden in den nächten Abschnitten einige Eigenschaften der Kettenbrüche erarbeiten, für eine ausführlichere Darstellung siehe zum Beispiel [4].

Algorithmus zur Berechnung der Kettenbruchentwicklung

Für ein gegebenes $x \in \mathbb{R}$ mit $x > 0$ suchen wir die Kettenbruchentwicklung (KBE), so dass $x = \langle q_0, q_1, \ldots, q_m \rangle$ gilt. Hier sollen, wie auch im Folgenden, die Werte q_i naturliche Zahlen sein. Dafür definieren wir folgenden iterativen Algorithmus:

1.13 Algorithmus. ***Kettenbruchentwicklung***

1. $q_0 = \lfloor x \rfloor \Rightarrow q_0 \in \mathbb{N}, q_0 \leqslant x < q_0 + 1$
2. $r_0 = x - q_0 \Rightarrow 0 \leqslant r_0 < 1$
3. *Wiederhole solange, bis* $q_m = q_i \in \mathbb{N}$*:*

3.1 $q_i = \lfloor \frac{1}{r_{i-1}} \rfloor$

3.2 $r_i = \frac{1}{r_{i-1}} - q_i$

4. Gebe $q_0, \ldots, q_m$ *aus.*

1.14 Beispiel. *Als Beispiel berechnen wir die KBE von* $x = \frac{4}{11}$*:*

$$\begin{aligned} q_0 &= 0 \\ r_0 &= \frac{4}{11} \\ q_1 &= \lfloor \frac{11}{4} \rfloor = 2 \\ r_1 &= \frac{3}{4} \\ q_2 &= \lfloor \frac{4}{3} \rfloor = 1 \\ r_2 &= \frac{1}{3} \\ q_3 &= 3 \\ &\Rightarrow \\ \frac{4}{11} &= \langle 0, 2, 1, 3 \rangle \\ &= 0 + \frac{1}{2 + \frac{1}{1 + \frac{1}{3}}} \end{aligned}$$

Eigenschaften von Kettenbrüchen

Die meisten der folgenden Eigenschaften von Kettenbrüchen sollten intuitiv verständlich sein. Die Beweise sind in den gängigen Lehrbüchern zur Zahlentheorie enthalten, z.B. in [2].

1. Wenn die KBE von x endlich ist, dann ist $x \in \mathbb{Q}$.
2. Wenn $x \in \mathbb{Q}$ ist, dann hat x eine endliche KBE.
3. Eine Zahl $x \in \mathbb{R} \setminus \mathbb{Q}$ hat genau dann eine periodische KBE, wenn es $a, b \in \mathbb{Q}$ gibt, so dass $x^2 + ax + b = 0$ gilt, wobei das Polynom $x^2 + ax + b \in \mathbb{Q}[x]$ irreduzibel ist. (D.h.: Genau dann, wenn x die Nullstelle eines irreduzibelen rationalen quadratischen Polynoms ist.)
4. Es gilt für $c \in \mathbb{N}$ dass
$$\langle q_0, \ldots, q_m \rangle > \langle q_0, \ldots, q_m + c \rangle$$
wenn m gerade ist und
$$\langle q_0, \ldots, q_m \rangle < \langle q_0, \ldots, q_m + c \rangle$$
wenn m ungerade ist.

Inverse Berechnung der Kettenbruchentwicklung

Nach dem wir einen Algorithmus zur Berechnung der KBE und einige Eigenschaften der KBE kennengelernt haben, stellt sich die Frage, wie man aus einer gegebenen KBE $x = \langle q_0, \ldots, q_m \rangle$ die Darstellung $x = \frac{n}{d}$ mit $n, d \in \mathbb{N}$ erhält. Ein erster naiver Ansatz wäre, den Kettenbruch komplett als Bruch $x = q_0 + \cfrac{1}{\ddots + \frac{1}{q_m}}$ aufzuschreiben und beginnend bei $q_{m-1} + \frac{1}{q_m}$ schrittweise in die gewünschte Darstellung zu transformieren. Es gibt allerdings einen effizienteren Algorithmus, der rekursiv zu jedem i die Darstellung $\frac{n_i}{d_i} = \langle q_0, \ldots, q_i \rangle$ liefert. Dafür benötigen wir:

1.15 Theorem. *Seien $x = \langle q_0, \ldots, q_m \rangle$ und $n_0 = q_0$, $n_1 = q_0 q_1 + 1$, $d_0 = 1$, $d_1 = q_1$ gegeben. Weiter sei*

$$\begin{aligned} n_i &= q_i n_{i-1} + n_{i-2} \\ d_i &= q_i d_{i-1} + d_{i-2}. \end{aligned}$$

Dann gilt:

$$\frac{n_i}{d_i} = \langle q_0, \ldots, q_i \rangle .$$

Außerdem gilt $\mathrm{ggT}(n_i, d_i) = 1$.

Beweis: Wir beweisen dieses Theorem mit Induktion.

Induktionsanfang: Es gelten

$$\frac{n_0}{d_0} = \frac{q_0}{1} = q_0 = \langle q_0 \rangle \qquad \surd$$

und

$$\frac{n_1}{d_1} = \frac{q_0 q_1 + 1}{q_1} = q_0 + \frac{1}{q_1} = \langle q_0, q_1 \rangle \qquad \surd$$

Induktionsannahme: Die Annahme gelte für Kettenbruchentwicklungen der Länge i.

Induktionsschritt: Es gilt:

$$\underbrace{\langle q_0, \ldots, q_i \rangle}_{Länge=i+1} = \underbrace{\left\langle q_0, \ldots, q_{i-1} + \frac{1}{q_i} \right\rangle}_{Länge=i} = \frac{n_i}{d_i}$$

Durch Anwenden der Induktionsannahme erhalten wir:

$$\begin{aligned}
\frac{n_{i-1}}{d_{i-1}} &= \frac{\left(q_{i-1}+\frac{1}{q_i}\right)n_{i-2}+n_{i-3}}{\left(q_{i-1}+\frac{1}{q_i}\right)d_{i-2}+d_{i-3}} \\
&= \frac{q_{i-1}n_{i-2}+n_{i-3}+\frac{n_{i-2}}{q_i}}{q_{i-1}d_{i-2}+d_{i-3}+\frac{d_{i-2}}{q_i}} \\
&= \frac{n_{i-1}+\frac{n_{i-2}}{q_i}}{d_{i-1}+\frac{d_{i-2}}{q_i}} \\
&= \frac{q_i n_{i-1}+n_{i-2}}{q_i d_{i-1}+d_{i-2}}
\end{aligned}$$

Es bleibt zu zeigen, dass der $\mathrm{ggT}(n_i, d_i) = 1$ ist, d.h. dass der Bruch $\frac{n_i}{d_i}$ gekürzt ist. Dazu betrachten wir folgende Matrizengleichung, die sofort aus den Rekursionsgleichungen folgt.

$$\begin{aligned}
\begin{pmatrix} n_i & d_i \\ n_{i-1} & d_{i-1} \end{pmatrix} &= \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} n_{i-1} & d_{i-1} \\ n_{i-2} & d_{i-2} \end{pmatrix} \\
&= \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} q_{i-1} & 1 \\ 1 & 0 \end{pmatrix}\cdots\begin{pmatrix} n_1 & d_1 \\ n_0 & d_0 \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} q_{i-1} & 1 \\ 1 & 0 \end{pmatrix}\cdots}_{i-1\,Faktoren}\begin{pmatrix} q_0q_1+1 & q_1 \\ q_0 & 1 \end{pmatrix}
\end{aligned}$$

Wenn man auf beiden Seiten der Gleichung die Determinanten der Matrizen bildet, erhält man:

$$n_i d_{i-1} - n_{i-1}d_i = (-1)^{i-1}\cdot 1$$

Aus obiger Gleichung folgt $n_i d_{i-1} \equiv \pm 1 \mod d_i$. Damit existiert aber $n_i^{-1} \mod d_i$ woraus folgt, dass $\mathrm{ggT}(n_i, d_i) = 1$ gilt. □

Für den obigen Beweis sei daran erinnert, dass sich die Determinante einer 2×2 Matrix folgendermaßen berechnen lässt:

$$\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - cb$$

Außerdem gilt $\det(AB) = \det(A)\det(B)$. Eine Matrix A ist genau dann invertierbar, wenn $\det(A) \neq 0$ gilt.

Man beachte, dass die Rekursionsgleichungen in Theorem 1.15 den Rekursionsgleichungen des Erweiterten Euklidischen Algorithmus gleichen. Außerdem erhält man für $q_i = 1$ die Fibonacci-Zahlen, als Nenner und Zähler der Näherungsbrüche.

1.16 Beispiel. *Das Verfahren soll an dem schon bekannten Beispiel* $x = \langle 0, 2, 1, 3\rangle$ *demonstriert werden:*

i	n_i	n_{i-1}	d_i	d_{i-1}	$\frac{n_i}{d_i}$
1	1	0	2	1	$\frac{1}{2} = \langle 0, 2\rangle$
2	1	1	3	2	$\frac{1}{3} = \langle 0, 2, 1\rangle$
3	4	1	11	3	$\frac{4}{11} = \langle 0, 2, 1, 3\rangle$

Approximieren mit Kettenbrüchen

Sei folgende Situation gegeben. Ein $x' \in \mathbb{Q}$ sei bekannt und weiterhin ist bekannt, dass für ein unbekanntes $x \in \mathbb{Q}$ gilt

$$x' = x(1 - \delta)$$

wobei $0 < \delta < 1$, $\delta \in \mathbb{Q}$. Der gegebene Bruch x' ist somit für kleines δ wenig kleiner als der unbekannte Bruch x. Der folgende Algorithmus berechnet, unter bestimmten Bedingungen die weiter unten erläutert werden, mit Hilfe der KBE von x' für ein ausreichend kleines δ den gesuchten Bruch x:

1.17 Algorithmus. ***Approximationsalgorithmus***
Wiederhole, bis x gefunden ist:

1. *Berechne die nächste Stelle q'_i der KBE von x'.*

2. *Berechne $a = \langle q'_o, \ldots, q'_i + 1\rangle$ wenn i gerade ist und $a = \langle q'_o, \ldots, q'_i\rangle$ wenn i ungerade ist. Damit gilt in beiden Fällen $a \geqslant x'$.)*

3. *Teste, ob $a = x$ gilt.*

Der letzte Schritt des Algorithmus setzt voraus, dass eine zusätzliche Eigenschaft von x bekannt ist, die x eindeutig als die gesuchte Lösung identifiziert. Wir werden im nächsten Abschnitt am Beispiel von RSA sehen, wie eine solche Überprüfung aussehen kann. Weiterhin behandeln wir im Abschnitt 1.7.2 für welche Größen von x und δ dieser Algorithmus funktioniert.

Angriff auf RSA mit kleinen privaten Exponenten

Nun haben wir also die Grundlagen für den Angriff auf RSA mit kleinem privaten Exponenten geschaffen. Sei $n = pq$ ein RSA-Modul mit p, q prim und e der öffentliche Exponent, d der geheime Exponent mit $ed \equiv 1 \mod \text{kgV}\,(p-1, q-1)$. Es existieren also $k, g \in \mathbb{N}$ mit $\text{ggT}(k, g) = 1$ und $ed = \frac{k}{g}(p-1)(q-1) + 1$. Wenn man auf beiden Seiten durch pqd teilt, erhält man:

$$\begin{aligned} \frac{e}{pq} &= \frac{k}{dgpq}(p-1)(q-1) + \frac{1}{dpq} \\ &= \frac{k}{dg}(1 - \delta) \end{aligned}$$

wobei

$$\delta \;=\; \left(\frac{p+q-1-\frac{g}{k}}{pq}\right)$$

Wenn δ ausreichend klein ist, liefert der Algorithmus 1.17 für $x' = e/pq$ und $x' = k/dg$ Kandidaten für k und dg. Wenn man die Gleichung $ed = \frac{k}{g}(p-1)(q-1)+1$ mit g multipliziert, erhält man

$$\underbrace{e}_{bekannt}\;\underbrace{(dg)}_{Kandidat} \;=\; \underbrace{k}_{Kandidat}\;(p-1)(q-1)+g$$

Daraus erhält man, unter der Annahme, das $g < k$ gilt, Kandidaten für $(p-1)(q-1)$

$$(p-1)(q-1) = \left\lfloor \frac{edg}{k} \right\rfloor .$$

Mit diesen Kandidaten lässt sich $p+q = -(p-1)(q-1)+n+1$ berechnen. Wenn man nun die quadratische Gleichung

$$x^2 - (p+q)x + n = (x-p)(x-q)$$

löst, erhält man Kandidaten für p und q. Wenn diese Teiler von n sind, ist man fertig. Andernfalls muss man den Algorithmus weiter durchführen.

1.18 Beispiel. *$n = pq = 8927$, $e = 2621$ und $x' = \frac{e}{n}$, $x = \frac{k}{dg}$*
$i = 0$)

$$q_0^{'} = \left\lfloor \frac{2621}{8927} \right\rfloor = 0, \quad a_0 =< q_0 + 1 >= 1 = \tfrac{1}{1} \quad \Rightarrow k = 1, dg = 1$$

aus $edg = k\left\lfloor \frac{edg}{k} \right\rfloor + g = edg + g$ folgt, dass $g = 0$ sein muss. Die Kandidaten sind also falsch, weil $(p-1)(q-1) = \left\lfloor \frac{edg}{k} \right\rfloor$ nicht 0 sein kann.
$i = 1$)

$$q_1^{'} = \left\lfloor \frac{8927}{2621} \right\rfloor = 3, \quad a_1 =< q_0, q_1 >= \tfrac{1}{3} \quad \Rightarrow k = 1, dg = 3$$

Aus dem selben Grund wie bei der letzten Iteration sind die Kandidaten falsch.
$i = 2$)

$$q_2^{'} = \left\lfloor \frac{2621}{1064} \right\rfloor = 2, \quad a_2 =< q_0, q_1, q_2 + 1 >=< 0, 3, 3 >= \tfrac{3}{10} \quad \Rightarrow k = 3, dg = 10$$

$$edg = k\left\lfloor \frac{edg}{k} \right\rfloor + g \Rightarrow 2621 \cdot 10 = 3 \cdot \left\lfloor \frac{2621 \cdot 10}{3} \right\rfloor + g \Rightarrow g = 26210 - 3 \cdot 8736 = 2$$

$$(p+q) = -\left\lfloor \frac{edg}{k} \right\rfloor + n + 1 = -8736 + 8927 + 1 = 192$$

Um p und q zu ermitteln muss folgende quadratische Gleichung gelöst werden:

$$x^2 - (p+q)x + n = 0$$

also

$$x^2 - 192x + 8927 = 0 \Rightarrow p = 113, q = 79$$

Wir haben also n in nur 3 Iterationen faktorisiert.

1.7.2 Warum und wann funktioniert der Algorithmus?

Sei

$$x^{'} = x(1-\delta) \quad x, x^{'} \in R$$

und weiter $x =< q_0, ..., q_m >$, dann liefert der Algorithmus x aus $x^{'}$ wenn

$$< q_0, ..., q_m - 1 > \leqslant x^{'} \leqslant < q_0, ..., q_m >$$

für m gerade bzw.

$$< q_0, ..., q_m + 1 > \leqslant x^{'} \leqslant < q_0, ..., q_m >$$

für m ungerade.

Was bedeutet das für δ?

1. Fall: $m = 0$
Aus $x^{'} = x(1-\delta)$ folgt $\delta = 1 - \frac{x^{'}}{x}$.
Es gilt $x =< q_0 >$ und $< q_0 - 1 > \leqslant x^{'} \leqslant < q_o >$.
Daraus folgt:

$$\delta = 1 - \frac{x^{'}}{x} \leqslant 1 - \frac{< q_0 - 1 >}{< q_0 >} = 1 - \frac{q_0 - 1}{q_0} = \frac{1}{q_0} = \frac{1}{n_0 d_0}$$

2.Fall: $m = 1$
Es gilt $x =< q_0, q_1 >$ und $< q_0, q_1 + 1 > \leqslant x' \leqslant < q_0, q_1 >$

$$\Rightarrow \delta = 1 - \frac{x'}{x} \leqslant 1 - \frac{< q_0, q_1 + 1 >}{< q_0, q_1 >} = 1 - \frac{q_0 + \frac{1}{q_1+1}}{q_0 + \frac{1}{q_1}} = 1 - \frac{q_0(q_1+1)+1}{q_1+1}\frac{q_1}{q_0q_1+1}$$

Auf einen Nenner gebracht ergibt sich dann:

$$\frac{(q_1+1)(q_0q_! + 1) - (q_0q_1 + q_0 + 1)q_1}{(q_! + 1)(q_0 + q_1 + 1)} = \frac{q_0q_1^2 + q_0q_1 + q_1 + 1 - q_0q_1^2 - q_0q_1 - q_1}{(q_1+1)(q_0q_1+1)}$$

$$= \frac{1}{(q_1+1)(q_0q_1+1)} = \frac{1}{(d_1+1)n_1} \geqslant \frac{1}{\frac{3}{2}d_1n_1},$$

da $d_1 \geqslant 2 \Rightarrow d_1 + 1 \leqslant \frac{3}{2}d_1$.

Der Algorithmus funktioniert also, wenn $\delta < \frac{3}{2}d_1n_1$.

3.Fall: $m \geqslant 2$, m gerade
Es gilt $x =< q_0, ..., q_m >$ und $x' > < q_0, ..., q_m - 1 >$

$$\Rightarrow \delta = 1 + \frac{x'}{x} \leqslant 1 - \frac{< q_0, ..., q_m - 1 >}{< q_0, ..., q_m >}$$

Für den Nenner gilt:

$$< q_0, ..., q_m >= \frac{n_m}{d_m} = \frac{q_mn_{m-1} + n_{m-2}}{q_md_{m-1} + d_{m-2}}$$

und für den Zähler:

$$< q_0, ..., q_m - 1 >= \frac{(q_m - 1)n_{m-1} + n_{m-2}}{(q_m - 1)d_{m-1} + d_{m-2}} = \frac{n_m - n_{m-1}}{d_m - d_{m-1}}$$

Daraus folgt für δ:

$$\delta \leqslant 1 - \frac{n_m - n_{m-1}}{d_m - d_{m-1}}\frac{d_m}{n_m} = \frac{(d_m - d_{m-1})n_m - (n_m - n_{m-1})d_m}{(d_m - d_{m-1})n_m}$$

$$= \frac{-d_{m-1}n_m + d_mn_{m-1}}{(d_m - d_{m-1})n_m} = \frac{(-1)^m}{(d_m - d_{m-1})n_m} = \frac{1}{(d_m - d_{m-1})n_m}$$

Es gilt : $d_{m-1}n_m - d_mn_{m-1} = -(-1)^m$. Das bedeutet, der Algorithmus funktioniert wenn:

$$\delta \leqslant \frac{1}{n_md_m}$$

4.Fall: $m \geqslant 2$, m ungerade
Es gilt:

$$x' > < q_0, ..., q_m + 1 >= \frac{(q_m+1)n_{m+1} + n_{m-2}}{(q_m+1)d_{m-1} + d_{m-2}} = \frac{n_m + n_{m-1}}{d_m + d_{m-1}}$$

$$\Rightarrow \delta \leqslant 1 - \frac{n_m + n_{m-1}}{d_m + d_{m-1}} \frac{d_m}{n_m} = \frac{1}{(d_m + d_{m-1})n_m}$$

Es $q_m \geqslant 2$ und $d_m = q_m d_{m-1} + d_{m-2}$

$$\Rightarrow d_m \geqslant 2 \cdot d_{m-1} + d_{m-2} \geqslant 2 \cdot d_{m-1}$$

$$\Rightarrow d_m + d_{m-1} \leqslant \frac{3}{2} d_m$$

Der Algorithmus funktioniert also, wenn

$$\delta \leqslant \frac{1}{\frac{3}{2} d_m n_m}$$

Betrachtet man alle 4 Fälle, so haben wir insgesamt folgenden Satz bewiesen.

1.19 Theorem. *Sei $x', x \in \mathbb{Q}$ mit*

$$x' = x(1 - \delta)$$

wobei $0 < \delta < 1$, $\delta \in \mathbb{Q}$. Dann liefert der Algorithmus 1.17 den richtigen Wert $x = n_m/d_m$ wenn

$$\delta \leqslant \frac{1}{\frac{3}{2} d_m n_m}$$

Es bleibt noch zu Untersuchen, für welche Wahl von Parametern RSA für diesen Angriff anfällig ist.

Was bedeutet das für RSA? Für RSA erhalten wir

$$ed = \frac{k}{g}(p-1)(q-1) + 1$$

mit $\mathrm{ggT}(k, g) = 1$

$$\Rightarrow \frac{e}{n} = \frac{k}{dg}(1 - \delta)$$

mit $\delta = \frac{p+q-1-\frac{g}{k}}{pq}$. Außerdem gilt $x = \frac{k}{dg} = \frac{n_m}{d_m}$. Für unser δ muß also gelten:

$$\delta < \frac{1}{\frac{3}{2} kdg}$$

Für große p,q folgt

$$\delta \approx \frac{p+q}{pq} < \frac{1}{\frac{3}{2} kdg} \Rightarrow dkg < \frac{pq}{\frac{3}{2}(p+q)}$$

Es gilt $kg < d$ und g kann als klein angenommen werden:

$$\Rightarrow d^2 g < \frac{pq}{\frac{3}{2}(p+q)}$$

und wenn p,q ungefähr gleich gross sind ($\approx \sqrt{n}$):

$$d^2 g < \frac{n}{\frac{3}{2}2\sqrt{n}} = \frac{\sqrt{n}}{3} \Rightarrow d < \frac{\sqrt[4]{n}}{\sqrt{3g}}$$

mit kleinem g führt dies zu folgender Faustregel: Der Angriff funktioniert, wenn die Bitlänge von d ein Viertel der Bitlänge von n beträgt. Die Laufzeit des Angriffs ist linear in der Bitlänge von n. Auch lange RSA Schlüssel mit 1024 Bit oder mehr lassen sich mit diesem Angriff faktorisieren, solange der private Schlüssel klein genug ist.

Kapitel 2

Faktorisierung großer Zahlen

In diesem Kapitel sollen Algorithmen vorgestellt werden, mit denen man zu einem gegebenen $n \in \mathbb{N}$ nicht triviale Teiler von n (also Teiler ungleich $1, n$) finden kann. Wir konzentrieren uns hierbei hauptsächlich auf den für RSA interessanten Fall, dass n das Produkt zweier Primzahlen ist. Die meisten der hier beschriebenen Algorithmen lassen sich aber für beliebige natürliche Zahlen verallgemeinern.

Um die Komplexität des Algorithmus abschätzen zu können, müssen einige Definitionen eingeführt werden.

2.1 Definition. *Für die Komplexität der von N abhängigen Algorithmen führen wir folgende Notation ein:*

$L_N(\alpha, c) := \exp\big((c+o(1))(\ln N)^{\alpha}(\ln\ln N)^{1-\alpha}\big)$ mit $0 \leqslant \alpha \leqslant 1$ und $c > 0$. $o(1)$ beschreibt das asymtotische Verhalten von N.

Wenn der zweite Parameter c ausgelassen wird, wird er mit $1/2$ angenommen.

α ist der wichtigere der beiden Parameter. In Abhängigkeit von α interpolliert $L_N(\alpha, c)$ zwischen der polynomiellen Komplexität für $\alpha = 0$ und der exponentiellen Komplexität für $\alpha = 1$. Für $\alpha < 1$ ist die Komplexität *subexponentiell*.

Eine positive Ganzzahl is dann B-glatt, wenn alle seine Primfaktoren $\leqslant B$ sind.

2.2 Lemma. *Seien $\alpha, \beta, r, s \in \mathbb{R}_{>0}$ mit $s < r \leqslant 1$.*

Dann ist eine zufällig gewählte positive Ganzzahl $\leqslant L_x(r, \alpha)$ mit einer Wahrscheinlichkeit von $L_x(r-s, -\alpha(r-s)/\beta)$ für $x \to \infty$
$L_x(s, \beta)$-glatt.

2.1 Probedivision

Das einfachste Verfahren ist die Probedivision. Hierbei wird versucht, die zu faktorisierende Zahl n durch alle möglichen Zahlen zu dividieren.

2.3 Algorithmus. ***Faktorisieren durch Probedivision***
Eingabe: n
Ausgabe: Ein Faktor von n, oder „n ist Primzahl“

1. *For t from 2 to $\lfloor\sqrt{n}\rfloor$*
   (a) *Falls $t \mid n$ (t teilt n): gebe t aus, stop*
2. *n ist eine Primzahl.*

Aufwand: Wenn p ungefähr so groß wie q ist, ist der Aufwand dieses Algorithmus $O(\sqrt{n})$. Für $n < 2^{16}$ ist dieses Verfahren unter Umständen die schnellste Faktorisierungsmethode. Eine kleine Optimierung ist im folgenden Abschnitt beschrieben.

2.1.1 Verbesserung der Probedivision durch das Sieb des Erasthotenes

Idee: Nach dem wir überprüft haben, dass n durch eine Zahl t *nicht* teilbar ist, wissen wir auch, dass kein Vielfaches von t n teilen kann. Wir können die Vielfachen einer bereits untersuchten Zahl durch eine Variante der Idee des Sieb des Erasthotenes ausschließen.

2.4 Algorithmus. ***Verbesserte Faktorisierung durch Probedivision***
Eingabe: n
Ausgabe: Ein Faktor von n, oder „n ist Primzahl“

1. *Array $E[0..\lfloor\sqrt{n}\rfloor]$; $E[t] = 0$ for t from 2 to $\lfloor\sqrt{n}\rfloor$*
2. *For t from 2 to $\lfloor\sqrt{n}\rfloor$*
   (a) *If $E[t] = 0$ then*
      i. *Falls $t \mid n$ (t teilt n): gebe t aus, stop*
      ii. *For s from t to $t\lfloor\sqrt{n}/t\rfloor$ step t do $E[s] = 1$*
3. *n ist eine Primzahl.*

2.2 $(p-1)$-Faktorisierungsalgorithmus

Diese Methode wurde 1974 von Pollard eingeführt. Nach dem kleinen Satz von Fermat gilt $a^{p-1} \equiv 1 \pmod{p}$ für eine Primzahl p und eine beliebige Ganzzahl a, die nicht durch p teilbar ist. Daraus folgt, dass wenn k ein Vielfaches von $p-1$ ist, dann ist auch $a^k \equiv 1 \pmod{p}$.

Desweiteren gilt: wenn $p \mid n$, dann $p \mid \gcd(a^k - 1, n)$.

Sei p unbekannt. Der obige Fakt kann benutzt werden, um den Primfaktor p von n zu berechnen. Hierzu prüfen wir, ob eine beliebige Ganzzahl $a \in [2, \ldots, n-1]$ mit $\gcd(a^k - 1, n)$

relativ prim zu n ist. Es ist möglich, allerdings sehr unwahrscheinlich, bereits durch die **gcd** Berechung einen Primfaktor von n zu finden.

Um diese Methode nutzen zu können, müssen wir ein k finden, wobei k ein Vielfaches von $p-1$ ist. Das Problem hierbei besteht darin, dass $p-1$ nicht im Vorraus bekannt ist. Wir gehen nun von der Annahme aus, dass n einen Faktor p hat, so dass $p-1$ aus kleinen Primfaktoren zusammengesetzt ist.

Wenn diese Annahme zutrifft können wir k als das Produkt von kleinen Primzahlpotenzen wählen. Die Anzahl der Primzahlpotenzen kann sehr groß werden, allerdings wird a^k rekursiv modulo n berechnet, so dass die Zahl k selbst nicht berechnet werden muss. Wenn der gerade beschriebene Ansatz fehlschlägt, wählen wir ein anderes k, das aus einer größeren Anzahl an Primzahlfaktoren besteht, und versuchen es erneut.

Das Ergebnis der beschriebenen Vorgehensweise ist, dass ein Primfaktor p von n in der Zeit gefunden werden kann, die proportional zum größten Primfaktor von $p-1$ ist. Es existiert auch eine direkte Methode, die anstelle von $p-1$ $p+1$ nutzt, wobei letzteres das zweite zyklotomische Polynom (Kreisteilungspolynom) ist. Das ist der Grund dafür, dass einige kryptographische Standards RSA-Moduli voraussetzen, die aus Produkten mehrerer Primzahlen bestehen, wobei $p+1$ und $p-1$ große Primzahlfaktoren haben.

Allerdings gibt es weitere Verfahren, die zyklotomische Polynome nutzen. Aufgrund dessen, aber hauptsächlich aufgrund der Faktorisierungsmethode auf Elliptischen Kurven, die nur von der Größe von p abhängt und eine wesentlich niedrigere Komplexität hat, ist die oben beschriebene Einschränkung der Primfaktoren überflüssig.

Idee: Finde ein $k \in \mathbb{N}$ und $a \in \mathbb{Z}_n^*$ so dass $a^k = 1 \bmod p$ gilt. Dann gilt

$$p \mid (a^k - 1)$$

Berechnet man nun

$$\mathrm{ggT}(a^k - 1, n) = g$$

dann gilt $g = p$, wenn $a^k \neq 1 \bmod q$. Damit hat man einen Faktor von n gefunden.

Problem: Wie findet man ein solches k?

Wenn $(p-1) \mid k$, dann gilt für alle $a \in \mathbb{Z}_n^*$

$$a^k = a^{t(p-1)} = \left(a^{(p-1)}\right)^t = 1^t = 1 \bmod p$$

Da man p und damit auch $p-1$ nicht kennt, nimmt man an, dass alle Primteilerpotenzen von $p-1$ kleiner als eine Schranke $B \in \mathbb{N}$ sind. *

Das bedeutet, wenn p_i prim und ${p_i}^{\alpha} \mid (p-1)$ und ${p_i}^{\alpha+1} \nmid (p-1)$ (Notation: ${p_i}^{\alpha} \,||\, (p-1)$), dann soll ${p_i}^{\alpha} \leqslant B$ gelten.

*Primzahlen p, für die $p-1$ nur kleine Primteiler hat, werden auch als glatt (engl. smooth) bezeichnet.

Sei $\alpha(p_i, B)$ die größte ganze Zahl, so dass ${p_i}^{\alpha(p_i,B)} \leqslant B$. Dann gilt für

$$k = \prod_{p_i \leqslant B} {p_i}^{\alpha(p_i,B)}$$

nach Voraussetzung: $(p-1) \mid k$. Somit liefert der Algorithmus unter der Bedingung, dass $a^k \neq 1 \bmod q$, für jedes $a \in \mathbb{Z}_n^*$ einen Teiler von n.

Eine sinnvolle Implementierung dieses Algorithmus startet mit einer kleinen Schranke B und erhöht diese, bis n erfolgreich faktorisiert ist.

Aufwand: Man kann zeigen, dass $k \approx e^B$. Das bedeutet, dass der Aufwand, den Wert a^k für ein $a \in \mathbb{Z}_n^*$ zu berechnen, $O(B)$ ist.

Der Aufwand dieser Methode ist stark davon abhängig, wie n gewählt wurde. Es gibt große RSA-Moduli, die mit dieser Methode schnell faktorisiert werden können. So wurde bis vor einigen Jahren zum Beispiel im BSI Algorithmenkatalog zur rechtsgültigen Digitalen Signatur festgelegt, dass $p-1$ und $q-1$ so gewählt sein müssen, dass die $p-1$ Methode nicht effizient sein kann. Diese Anforderung wurde aber nach der Entwicklung der Faktorisierungs-Methode mit Elliptischen Kurven fallen gelassen.

2.3 Fermat-Morrison-Brillharts Vorgehen

Auch hier sei $n = pq$ und p, q prim, die Idee ist aber allgemeiner und funktioniert ohne Änderungen auch für aus mehreren Primzahlpotenzen zusammengesetzte Zahlen.

2.3.1 Fermats Idee

Idee: Finde $x, y \in \mathbb{Z}_n^*$, so dass

$$x^2 \equiv y^2 \pmod{n} \quad \text{and} \quad \gcd(xy, n) = 1 \ . \tag{2.1}$$

Daraus folgt

$$x^2 - y^2 = (x-y)(x+y) = 0 \bmod n$$

und $\mathrm{ggT}(x \pm y, n)$ liefert (mit großer Wahrscheinlichkeit) einen nichttrivialen Faktor von n.

2.3.2 Random Squares

2.5 Algorithmus. ***Random Squares***

Eingabe: n

Ausgabe p, q

1. *Wähle* $x \in_R \mathbb{Z}_n^*$

2. *Berechne* $x^2 \bmod n$

3. *Teste, ob* $x^2 \bmod n$ *ein Quadrat in* $\mathbb{N}$ *ist*

   (a) *Wenn ja, berechne* y *mit* $y^2 = x^2 \bmod n$ *und* $p = \mathrm{ggT}(x - y, n)$*, gebe* p*,* $q = \frac{n}{p}$ *aus*

   (b) *Wenn nein, wähle neues* x

Unter der Annahme, dass für $x \in_R \mathbb{Z}_n^*$ die Werte $x^2 \bmod n$ zufällig in $\mathbb{Z}_n^*$ verteilt sind, ist die Wahrscheinlichkeit, dass $x^2 \bmod n$ eine Quadratzahl in $\mathbb{N}$ ist, ungefähr $\frac{1}{\sqrt{n}}$, d.h. der Aufwand dieses Algorithmus ist $O(\sqrt{n})$.

2.3.3 Erste Verbesserung

Man sucht x, derart dass $x^2 \bmod n$ klein ist. Also ist die Wahrscheinlichkeit, dass $x^2 \bmod n$ ein Quadrat ist höher. Eine gute Strategie besteht darin, Zahlen der Form

$$x = \lceil \sqrt{k\,n} \rceil + j \qquad \text{mit kleinen } k, j \in \mathbb{N}$$

zu nutzen. Dann ist

$$x^2 \approx k\,n + 2\sqrt{k\,n}\,j + j^2$$

und

$$x^2 \bmod n \approx 2\sqrt{k\,n}\,j + j^2 \ .$$

Eine Implementierungsmöglichkeit ist, die obere Schranke für k und j allmählich zu erhöhen. Wir werden aber noch bessere Methoden sehen, um x zu wählen, bzw. konstruieren, so dass $x^2 \bmod n$ klein ist.

2.3.4 Faktorisieren mit Random Squares a la Dixon

Man kann den Random-Squares-Algorithmus verbessern, indem man die in allen Durchläufen, in denen $x^2 \bmod n$ kein Quadrat in $\mathbb{N}$ ist, gewonnenen Informationen nicht verwirft, sondern sammelt.

Die Idee von Dixon ist die folgende: Wähle $x < n$ zufällig und speichere die x-Werte, für welche $x^2 \bmod n$ über einer kleinen Menge Primzahlen (der Faktorbasis) zerfällt. Wir erhalten hierraus die Relationen $x_i^2 \equiv z_i \bmod n$. Diese können wir passend miteinander multiplizieren und erhalten somit:

$$\Big(\prod_{\text{ausgewählte } i} x_i\Big)^2 \equiv Z$$

wobei $z = \prod_{\text{ausgewählte } i} z_i \equiv Z$ ein Quadrat in $\mathbb{Z}$ ist.

Das ist die Grundidee für den folgenden Algorithmus.

2.6 Algorithmus. ***Random Squares Improved***

Eingabe: n
Ausgabe: Faktor von n

1. Wähle eine Menge $\mathcal{B}$ von Primzahlen (Faktorbasis), z. B. $\mathcal{B} = \{pPrimzahl\ mit$p¡B$\}$. Sei $s = |B|$.
2. Wähle $x_i \in_R \mathbb{Z}_n^*$
   (a) Berechne $z_i = {x_i}^2 \bmod n$
   (b) Teste, ob z_i über $\mathcal{B}$ zerfällt, d.h., ob alle Primteiler von z_i in B enthalten sind
   (c) Wenn z_i über $\mathcal{B}$ zerfällt, dann bestimme $\alpha_{i,j} \in \mathbb{N}$, so dass
   $$z_i = \prod_{p_j \in B} {p_j}^{\alpha_{i,j}}$$
   und berechne (und speichere) den Vektor
   $$\alpha_i = (\alpha_{i,1} \bmod 2, \alpha_{i,2} \bmod 2, \ldots, \alpha_{i,j} \bmod 2) \in \mathbb{F}_2^s$$
3. Wähle geeignete z_i, so dass die Vektoren α_i linear abhängig sind, berechne $z = \prod_{ausgewählte\, i} z_i$ und $x \prod_{ausgewählte\, i} x_i$
4. Berechne $t = \mathrm{ggT}(x - \sqrt{z}, n)$ und gebe t aus

---

Algorithm 1. Fermat–Morrison–Brillhart factoring algorithm

INPUT: A rational integer n.

OUTPUT: A nontrivial factor of n.

---

1. ***Choose a factor base***
   Given a smoothness bound B, let the $factor\ base\ \mathcal{P}$ be the set of primes $\leqslant B$.
   The cardinality of $\mathcal{P}$ is $\pi(B)$.
2. ***Collecting relations***
   Collect more than $|\mathcal{P}|$ integers v such that $v^2 \bmod n$ is B-smooth:
   $$v^2 \bmod n = \left(\prod_{p \in \mathcal{P}} p^{e_{v,p}} \right).$$
   These congruences are called the $relations$.
   Let $\mathcal{V}$ be the resulting set of relations of cardinality $|\mathcal{V}| > |\mathcal{P}|$.
3. ***Linear algebra***
   Each $v \in \mathcal{V}$ gives rise to a $|\mathcal{P}|$-dimensional vector $(e_{v,p})_{p \in \mathcal{P}}$. Since $|\mathcal{V}| > |\mathcal{P}|$, the vectors $\{(e_{v,p})_{p \in \mathcal{P}} : v \in \mathcal{V}\}$ are linearly dependent. This implies that there exist at least $|\mathcal{V}| - |\mathcal{P}|$ linearly independent subsets $\mathcal{S}$ of $\mathcal{V}$ for which $\sum_{v \in S} e_{v,p} = 2s_p$ is even for all $p \in \mathcal{P}$. These subsets $\mathcal{S}$ can, in principle, be found using Gaussian elimination modulo 2 on the matrix having the vectors $(e_{v,p})_{p \in \mathcal{P}}$ as rows.

4. ***Compute a solution***
For any subset $\mathcal{S}$ of $\mathcal{V}$ given by the linear algebra stage and the corresponding integer vector $(s_p)_{p\in\mathcal{P}}$, the integers
$$x = \prod_{v\in S} v \mod n \quad \text{and} \quad y = \prod_{p\in\mathcal{P}} p^{s_p} \mod n$$
form a solution to the congruence $x^2 \equiv y^2 \pmod{n}$. Each of the $|\mathcal{V}| - |\mathcal{P}|$ independent subsets thus leads to an independent chance of at least 50% to produce a nontrivial factor of n.

Bemerkung: z_i ist ein Quadrat in $\mathbb{N}$ genau dann, wenn $\alpha_i = (0,\ldots,0)$, d.h. wenn alle $\alpha_{i,j} = 0 \bmod 2$ sind. In diesem Fall ist
$$y_i = \prod_{p_j\in B} {p_j}^{\alpha_{i,j}/2}$$
eine Quadratwurzel von z_i, denn
$${y_i}^2 = \left(\prod_{p_j\in B} {p_j}^{\alpha_{i,j}/2}\right)^2 = \prod_{p_j\in B} {p_j}^{\alpha_{i,j}} = z_i$$

Der Algorithmus liefert eine Menge von Vektoren $\alpha_i \in \mathbb{F}_2^s$. Wenn diese Menge linear abhängig ist, d.h., $\exists\lambda_i \in \mathbb{F}_2$, so dass $\sum \lambda_i\alpha_i = (0,\ldots,0)$ gilt, dann berechnet man
$$x = \prod_i {x_i}^{\lambda_i}$$
Damit gilt
$$\begin{aligned} x^2 &= \left(\prod_i {x_i}^{\lambda_i}\right)^2 \\ &= \prod_i \left({x_i}^2\right)_i^{\lambda} \\ &= \prod_i {z_i}^{\lambda_i} \bmod n \end{aligned}$$
Es gilt $z_i = \prod_{p_j\in B} {p_j}^{\alpha_{i,j}}$. Damit ergibt sich
$$\begin{aligned} x^2 &= \prod_i \left(\prod_{p_j\in B} {p_j}^{\alpha_{i,j}}\right)^{\lambda_i} \\ &= \prod_{p_j\in B}\prod_i \left({p_j}^{\alpha_{i,j}}\right)^{\lambda_i} \\ &= \prod_{p_j\in B} {p_j}^{\sum_i \lambda_i\alpha_{i,j}} \end{aligned}$$

$\sum_i \lambda_i \alpha_{i,j} = 0 \bmod 2$ für alle j, da $\sum_i \lambda_i \alpha_{i,j} = (0, \ldots, 0)$. Das bedeutet

$$x^2 = \left(\prod_{p_j \in B} {p_j}^{(\sum_i \lambda_i \alpha_{i,j})/2} \right)^2$$

Damit erfüllt

$$y = \prod_{p_j \in B} {p_j}^{(\sum_i \lambda_i \alpha_{i,j})/2}$$

die gewünschte Gleichung $x^2 = y^2 \bmod n$.

2.7 Beispiel. *$n = 3233$ und $B = \{2, 3, 5, 7, 11\}$*

- *$x_1 = 1735$ und ${x_1}^2 \bmod n = 302$*
  Testen, ob x_1 über B zerfällt: $302 = 2 \cdot 151$
  $\Rightarrow$ x_1 zerfällt nicht über B
- *$x_2 = 373$ und ${x_2}^2 \bmod n = 110 = 2 \cdot 5 \cdot 11$ zerfällt über B und*
  $\alpha_2 = (1, 0, 1, 0, 1)$
- *$x_3 = 256$ und ${x_3}^2 \bmod n = 876 = 2^2 \cdot 3 \cdot 73$*
  $\Rightarrow$ x_3 zerfällt nicht über B
- *$x_4 = 1863$ und ${x_4}^2 \bmod n = 1760 = 2^5 \cdot 5 \cdot 11$ und*
  $\alpha_4 = (1, 0, 1, 0, 1)$

Es gilt $\alpha_2 + \alpha_4 = (0, 0, 0, 0, 0)$, d.h.,

$$x = x_2 \cdot x_4 = 373 \cdot 1836 = 694899 \qquad \text{und} \qquad y = \sqrt{z_2 \cdot z_4} = 440$$

Ein Faktor von n ergibt sich damit zu $\mathrm{ggT}(x - y, n) = 53$

2.3.5 Continued fraction method

Continued fraction method (Abgekürzt CFRAC) ist die Methode, die ursprünglich, im Jahre 1970, von Morrison und Brillhart genutzt wurde, um die siebte Fermatzahl (39 Ziffern) zu faktorisieren.

CFRAC sucht nach Kongruenzen $x^2 \equiv r \pmod{n}$, wobei r klein ist, für die $r = O(\sqrt{n})$ gilt. Für jede gefundene Kongruenz wird die Faktorisierung von r über die Faktorbasis probiert. Wenn r glatt ist, wird die Kongruenz gespeichert. Die gespeicherten Kongruenzen werden nun so miteinander multipliziert, dass sich auf beiden Seiten der Gleichung Quadrate ergeben. Falls n ein perfektes Quadrat ist, ist es einfach dieses zu faktorisieren. Diesen Fall schließen wir allerdings in Anbetracht der Anmerkung zu Beginn des Abschnitts aus.

Es existieren unendlich viele gebrochenrationale Approximationen p/q für $\sqrt{n}$, so dass gilt:

$$\left| \frac{p}{q} - \sqrt{n} \right| < \frac{1}{q^2},$$

Solche Approximationen erhalten wir durch die (konvergenten) Teilbrüche der (unendlichen) Kettenbruchentwicklung von $\sqrt{n}$. Wenn p/q so eine Approximation ist und ϵ ist der Form, dass $p/q = \sqrt{n} + \epsilon/q^2$ gilt, dann gilt auch:

$$p^2 - nq^2 = \left(q\sqrt{n} + \epsilon/q\right)^2 - nq^2 = 2\epsilon\sqrt{n} + \epsilon^2/q^2.$$

Da $|\epsilon| < 1$, erhalten wir $|p^2 - nq^2| < 2\sqrt{n} + 1/q^2$, wobei all die Werte von $p^2 - nq^2$ die gewünschte Komplexität $O(\sqrt{n})$ haben.

2.8 Bemerkung.

Man beachte, dass der Zähler und der Nenner p und q durch die Rekursionformeln gegeben sind, die die Koeffizienten der Kettenbruchentwicklung von $\sqrt{n}$ beinhalten. Da die Kettenbruchentwicklung periodisch ist, können p und q direkt modulo n berechnet werden, anstatt mit Langzahl-Integern zu rechnen zu müssen. Diese Vorgehensweise wurde von Montgomery vorgeschlagen.

Man beachte weiterhin, dass die Faktorbasis in Wirklichkeit nur halb so groß ist, wie sie sein könnte: Wenn $p' \mid (p^2 - nq^2)$, dann muss n ein quadratischer Rest modulo p' sein, solange $p' \mid n$. Da wir annehmen, dass n kein perfektes Quadrat ist, haben in einer asymptotischen Betrachtung nur die Hälfte der Primzahlen, die n als quadratischen Rest haben. Aus dieser Betrachtung folgt, dass wenn B die Glattheitsgrenze ist, die Kardinalität der Faktorbasis $\approx \pi(B)/2$ sein muss.

2.4 Das Quadratische Sieb

Problem: Die Random Square Methode benötigt die meiste Zeit, um Zahlen z_i durch die Primzahlen $p_i \in B$ zu dividieren, was in den meisten Fällen jedoch erfolglos ist.

Idee: Um den Algorithmus zu beschleunigen, braucht man ein Kriterium, das die Anzahl der erfolglosen Divisionen minimiert.

Dazu wählt man zwei Schranken $P, A \in \mathbb{N}$, so dass $P < A < P^2$ (z.B. $P \approx e^{\sqrt{\log n \log\log n}}$) und setzt

$$S = \left\{ \lfloor\sqrt{n}\rfloor + 1, \lfloor\sqrt{n}\rfloor + 2, \ldots, \lfloor\sqrt{n}\rfloor + A \right\}$$

Anschließend berechnet man alle Werte $t^2 - n$ für $t \in S$. Man beachte, dass $t^2 - n$ der kleinste positive Rest von t^2 modulo n ist, wenn A hinreichend klein gewählt wurde.

Als Faktorbasis verwendet man

$$B = \left\{ p \in \mathbb{N} \mid p \text{ prim }, p \leqslant P, \left(\frac{n}{p}\right) = 1 \right\} \cup \{2\}$$

wobei $\left(\frac{n}{p}\right) = 1$ bedeutet, dass n ein Quadrat modulo p ist, d.h. es gibt ein y, so dass $y^2 = n \bmod p$ gilt. Dies begründet sich darin, dass $p \mid (t^2 - n)$ äquivalent zu $t^2 = n \bmod p$

ist und damit gilt, dass n ein Quadrat modulo p ist. Das bedeutet, dass alle Zahlen $t^2 - n$ teilerfremd zu p sind, wenn n kein Quadrat modulo p ist.

Im weiteren Vorgehen dividiert man für $p = 2$ alle Elemente $t^2 - n$ durch die höchste 2er-Potenz, die $t^2 - n$ teilt. Man notiert den jeweils größten Exponenten in der entsprechenden Spalte in folgender Tabelle.

t	$t^2 - n$	2	3	...	P
$1 + \lfloor\sqrt{n}\rfloor$		1			
$2 + \lfloor\sqrt{n}\rfloor$		0			
$\vdots$		$\vdots$			
$A + \lfloor\sqrt{n}\rfloor$					

Für alle $p \in B$ mit $p \neq 2$ geht man folgendermaßen vor:

Man löst die Gleichung

$$t^2 = n \bmod p^\beta$$

für steigendes β so lange, bis diese keine Lösung mehr in S hat.

Seien t_1 und t_2 die Lösungen der Gleichung für das maximale β. Insbesondere ist

$${t_1}^2 - n = 0 \bmod p$$

Daraus folgt

$$(t_1 + p)^2 - n = 0 \bmod p$$

und weiter

$$(t_1 + cp)^2 - n = 0 \bmod p$$

Allgemein folgt aus ${t_1}^2 - n = 0 \bmod p^\alpha$

$$(t_1 + cp^\alpha)^2 - n = 0 \bmod p^\alpha$$

Den maximalen Wert von β schreibt man in obiger Tabelle in die entsprechende Spalte für $p \in B$ in die Zeile bei t_1. Ausgehend von t_1 geht man in der Tabelle ein Vielfaches von p nach oben bzw. unten und trägt in diesen Zeilen in der Spalte von p

- eine 1, wenn der Abstand von t_1 durch p^1, aber nicht mehr durch p^2 teilbar ist
- eine 2, wenn der Abstand von t_1 durch p^2, aber nicht mehr durch p^3 teilbar ist, usw.

Anschließend füllt man die Zeilen ausgehend von t_2 analog aus.

Die Zahlen $t^2 - n$, die über der Faktorbasis zerfallen, lassen sich jetzt leicht erkennen, da diese in der entsprechenden Zeile für t in der Spalte für $t^2 - n$ eine 1 stehen haben, d.h. sich vollständig in ihre Primfaktoren bzw. Primfaktorpotenzen aus B haben zerlegen lassen.

Eine geschickte Implementierung dividiert die Werte $t^2 - n$ bereits während des beschriebenen Schleifendurchlaufs durch die entsprechenden Primfaktorpotenzen.

Das weitere Vorgehen entspricht dem des Random Square Algorithmus. Das bedeutet, dass für die Zahlen, welche über der Faktorbasis zerfallen, der Vektor der Exponenten ermittelt wird. Mit Hilfe einer linear abhängigen Menge dieser Vektoren wird der gesuchte Wert y ermittelt, so dass $y^2 = x^2 \bmod n$ gilt.

2.5 Extra Material für das Quadratische Sieb

Das Quadratische Sieb

Bei CFRAC wird viel Zeit für das Testen der Reste $p^2 - nq^2$ auf B-Glattheit verbraucht, da diese quasi per Bruteforce faktorisiert werden. Als Abhilfe können Pollard's $p-1$, rho Methoden und letztendlich auch ECM genutzt werden.

Das von Pomerance [**?**, **?**] vorgeschlagene Quadratische Sieb entfernt diese Bürde fast vollständig. Es versucht geleichermaßen *kleine* quadratische Reste zu finden. Die so durch das Quadratische Sieb gefundenen Reste sind geringfügig größer als die Reste, die über CFRAC gefunden werden, können aber in der Praxis wesentlich schneller getestet werden.

Die Methode basiert auf der folgenden Beobachtung: Wenn $f(X) \in \mathbb{Z}[X]$ ein Polynom mit ganzzahligen Koeffizienten ist und p eine Primzahl, dann gilt $f(i) \equiv f(i+p) \pmod{p}$ für alle i.

erste Idee: Angenommen, wir wollen die aufeinanderfolgenden Werte i eines Polynoms f, auf Glattheit testen, wobei gilt $i \in [0, \ldots, L-1]$. Im Folgenden wird die erste Idee beschrieben, die dies ermöglicht.

Als erstes erstellen wir eine Wertetabelle für alle $f(i)$, für die gilt $0 \leqslant i < L$. Für jede Primzahl p in der Faktorbasis und für jedes i, für welches die Bedingung $f(i) \equiv 0 \pmod{p}$ erfüllt ist, tauschen wir unseren Tabelleneintrag $f(i)$ mit $f(i)/p^{e(i)}$ aus. Dabei stellt $p^{e(i)}$ den größten Exponenten von p dar, der $f(i)$ teilt.

Dies wird erreicht durch die Berechnung der Wurzeln r von $f(X)$ modulo p mit $0 \leqslant r < p$ und anschließender Division von $f(r+kp)$ durch die höchste Potenz von p, bei der für all diese Wurzeln und für alle ganzzahligen $k \geqslant 0$ gilt, dass $r + kp < L$. Nachdem wir alle Primzahlen in unsere Faktorbasis umgewandelt haben, testen wir, ob ein Tabelleneintrag ± 1 ist. Wenn dies zutrifft, war das entsprechende $f(i)$ glatt. Allerdings müssen dafür zu viele Berechnungen und Divisionen durchgeführt werden. Eine alternative Methode ist die folgende.

Algorithm 2. Sieve method

INPUT: A polynomial $f(X) \in \mathbb{Z}[X]$, an integer L and a smoothness bound B.
OUTPUT: A list $\mathcal{L}$ of values i with $0 \leqslant i < L$ for which $f(i)$ is B-smooth.

1. **for** $i = 0$ **to** $L-1$ **do** $s_i \leftarrow 0$ [the s_i correspond to the $f(i)$]
2. **for** all $p \leqslant B$ **do** [i.e. in the factor base]
3. **for** all roots r of $f(x)$ modulo p with $0 \leqslant r < p$ **do**
4. **for** all $k \geqslant 0$ with $r + kp < L$ **do**
5. $s_{r+kp} \leftarrow s_{r+kp} + \ln p$
6. **for** $i = 0$ **to** $L-1$ **do**
7. **if** s_i is "close" to $\ln f(i)$ **then if** $f(i)$ is B-smooth **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \{i\}$.

8. **return** $\mathcal{L}$

In der Praxis sind die Sieb-Werte s_i und die Werte $\ln p$ gerundete Ganzzahlen und hängen von der gewählten Basis ab (diese kann 2 oder eine Potenz von 2 sein). Wir vertiefen uns nicht in die Details, wie geprüft werden kann, ob s_i nah an $\ln f(i)$ liegt. Wichtig ist, dass für gewöhnlich eine ausreichend gute Approximation (z.B. durch eine lineare Funktion) für den Bereich der Sieb-Werte gefunden werden kann. Die Werte s_i nach dem Schritt 5 werden *residual logarithms* genannt.

Sei $v(i) = i + \lfloor\sqrt{n}\rfloor$ für kleine i, dann ist $v(i)^2 \bmod n = (i + \lfloor\sqrt{n}\rfloor)^2 - n \approx 2i\sqrt{n}$. Die obige Sieb-Methode wird angewandt auf $f(i) := v(i)^2 - n$.

Mit $B = L_n[1/2, 1/2]$ und unter der Annahme, dass $\big(v(i)^2 \bmod n\big)$ sich als eine Zufallszahl nahe an $\sqrt{n} = L_n[1, 1/2]$ verhält, könnten wir aus dem Lemma 2.2 folgern, dass es mit der Wahrscheinlichkeit $L_n[1/2, -1/2]$ B-glatt ist.

Diese Annahme ist offensichtlich falsch. Wenn eine ungerade Primzahl p $\big(v(i)^2 \bmod n\big)$ teilt, aber n nicht teilt, dann ist $\big(i+\lfloor\sqrt{n}\rfloor\big)^2 \equiv n \pmod p$, so dass n ein quadratischer Rest modulo p ist (die Geschichte wiederholt sich) und die Kardinalität der effektiven Faktorbasis $\approx \pi(B)/2$ ist. Andererseits erwarten wir für jedes in Frage kommende p zwei Wurzeln von $f(X)$ modulo p. Das Ergebnis dieser Überlegung ist, dass die Wahrscheinlichkeit der Glattheitseigenschaft, sehr nah an der aprioren, möglicherweise naiven, Vermutung liegt.

Sei $\mathcal{P}$ die gewählte Faktorbasis. Da wir mehr als $|\mathcal{P}| = L_n[1/2, 1/2]$ Relationen finden müssen, müssen

$$\frac{L_n[1/2, 1/2]}{L_n[1/2, -1/2]} = L_n[1/2, 1] = O\Big(e^{(1+o(1))\sqrt{\ln n \ln\ln n}}\Big)$$

unterschiedliche i's gesiebt werden. Das rechtfertigt die Annahme, dass i klein ist. Pomerance's zeigte, dass die erwartete (heuristische) Gesamtdauer des Quadratischen Siebes $L_n[1/2, 1]$ beträgt.

Um die Performance dieses Algorithmus zu verbessern, können verschiedene Ansätze genutzt werden, die in den nachfolgenden Abschnitten beschrieben werden.

Multiple polynomials

Wir haben gesehen, dass $L_n[1/2, 1]$ unterschiedliche $f(i)$-Werte gesiebt werden müssen. Die Auswirkung großer i's ist klar erkennbar, weil je größer i wird, desto mehr nimmt der Anteil an B-glatten Zahlen ab. Davis und Holdridge schlugen vor, mehr Polynome zu nutzen. Zeitgleich zu ihnen wurde eine viel praktischere Lösung von Montgomery und anderen Autoren vorgeschlagen. Im Folgenden werden diese Vorschläge betrachtet.

Montgomery schlug vor, quadratische Polynome der Form $f(X) = a^2X^2 + bX + c$ zu nutzen, wobei $b^2 - 4a^2c = kn$ mit $k = 1$ falls $n \equiv 1 \pmod 4$ und $k = 4$ falls $n \equiv 3 \pmod 4$. Man beachte, dass

$$f(X) = \left(aX + \frac{b}{2a}\right)^2 - \frac{b^2 - 4a^2c}{4a^2} \equiv \left(aX + \frac{b}{2a}\right)^2 \pmod n.$$

Somit ergibt sich, dass wenn a, b, und c klein genug sind, kann erwartet werden, dass diese Polynome genauso nützlich sind wie $v(x)^2$.

Um die Werte von $f(X)$ in dem Sieb-Interval $\mathcal{I}$ so klein wie möglich zu machen, sagen wir $2m$ lang, initial wollen wir das Interval um das Minimum der Funktion herrum legen welches wir über die Beziehung $X = \xi := -b/(2a^2)\cdot$ erhalten. Somit ergibt sich $\mathcal{I} = [\xi - m, \xi + m]$. Wir wollen, dass $|f(\xi)| \approx |f(\xi + m)|$ der Bedingung $b^2 - 4a^2c = kn$ unterliegt. Das hat zur Folge, dass wir den Fall, dass $f(\xi)$ und $f(\xi + m)$ das gleiche Vorzeichen haben, ausschließen. Somit erhalten wir im Falle, dass $(\xi) < 0 < f(\xi + m)$

$$a^2 \approx \frac{\sqrt{kn/2}}{m},\ b \approx 0 \ \text{ and } \ c \approx -m\sqrt{kn/8} < 0.$$

Der Wert des größten Polynoms ist in etwa $|c|$, das maximal $m\sqrt{n/2}$ sein kann. Um die L-Werte zu sieben, können wir $L/(2m)$ unterschidliche Polynome benutzen, die $2m$ Werte pro Polynom sieben. Folglich ist der größte Rest in etwa $O\big(m\sqrt{n}\big)$ anstatt $O(L\sqrt{n})$, wie es im Standardalgorithmus der Fall ist.

Frage: wie wählen wir a, b, und c?
Als erstes wird eine ungerade Primzahl für a ausgewählt, (mit $a \approx (kn/2)^{1/4}m^{-1/2}$), so dass kn ein quadratischer Rest modulo a ist. Als nächstes rechnen wir die Gleichung $b_0^2 \equiv kn \pmod{a}$ für b_0 und die Gleichung $(b_0 + \ell a)^2 \equiv kn \pmod{a}^2$ für ℓ. Im nächsten Schritt setzen wir $b = b_0 + \ell a$ oder $b = b_0 + \ell a - a^2$, je nach dem, welches die gleiche Parität hat wie kn. Und zu aller Letzt setzen wir $c = (b^2 - kn)/(4a^2)$, das nach Konstruktion eine Ganzzahl ist, und $b^2 - 4ac^2 = kn$.

Andere Autoren empfehlen, Polynome der Form $f(X) = aX^2 + 2bX + c$ mit $a > 0$ zu nehmen, mit $\delta := b^2 - ac > 0$, wobei $n \mid \delta$. Diese Polynome sind gleichermaßen brauchbar als $v(x)^2$, da

$$af(X) = (aX + b)^2 - (b^2 - ac) \equiv (aX + b)^2 \pmod{n}.$$

Die daraus resultierende Methode wird "Multiple Polynomial Quadratic Sieve" (oder MPQS) genannt. Es gibt unterschiedliche Wege, um diesen Standardalgorithmus zu verbessern. Im Folgenden werden die wichtigsten und allgemeingültigen Ideen vorgestellt.

Self-initializing polynomials

Es liegt die Vermutung nahe, dass man versuchen sollte, die Polynome so oft wie möglich zu wechseln, um die Reste so klein wie möglich zu machen. Dies kann allerdings zu Kollisionen führen (i. e. zwei gleiche kleine glatte Reste zu finden). Außerdem müssen, gerade bei moderater Umschaltung, für jedes Polynom alle Wurzeln modulo aller Primzahlen $\leqslant B$ gerechnet werden. Ein anderer sehr zeitintensiver Teil ist die Inversion des führenden Koeffizienten a^2 modulo jeder Primzahl in der Faktorbasis.

Self-initialization (Eigeninitialisierung) funktioniert wie folgt: man wählt ein nicht primes a, dafür aber das Produkt einiger weniger ($t \approx 10$) mittelgroßer Primzahlen p mit $\left(\frac{n}{p}\right) = 1$. Die Anzahl der möglichen b, und somit die Anzahl der möglichen Polynome mit dem führenden Term a, ist gleich der Anzahl der Lösungen von $b_0^2 \equiv kn \pmod{a}$ oder $b^2 \equiv n \pmod{a}$

im Fall von Cohen bzw. Montgomery, was gleich 2^{t-1} ist. Dieses Verfahren führt zu einem Geschwindigkeitsgewinn um etwa Faktor 2^{t-1} im zeitintensivsten Fall der Wurzelberechnung während der Sieb-Initialisierungsphase. Schnelle Wurzelberechnung ermöglicht es, vom Polynomwechel wesentlich mehr zu profotieren, und somit das Sieben kleiner Reste, die mit höherer Wahrscheinlichkeit glatt sind. Als Gesamtergebnis erhalten wir einen Geschwindigkeitsgewinn um etwa Faktor 2.

Large prime variations

Das Siebverfahren des Algorithmus, der unter 2 beschrieben ist, sucht nach Werten von i, so dass $f(i)$ B-glatt ist. Der Algorithmus kann nun so angepasst werden, dass er zusäzlich auch Werte für i finden kann, für welche $f(i)$ eine B-glatte Zahl multipliziert mit einer oder mehreren Primzahlen ist, die nicht wesentlich Grösser als B sind. Dies kann erreicht werden, indem der Schwellwert bei der Betrachtung der Logartihmen gesenkt wird. Diese zusätzliche Primzahl in der Faktorisierung von $f(i)$ wird *große Primzahl* genannt. Wenn wir zwei Werte i finden, für welche $f(i)$ die gleichen *großen Primzahlen* hat, dann können die entsprechenden Kongruenzen (*Teilrelationen*) miteinander multipliziert oder dividiert werden, um eine "neue" einfache Relation *(volle Relation)* zu erhalten, die im weiteren Verlauf des Algorithmus verwendet werden kann.

Kombinierte Teilrelationen ergeben somit eine weniger spärlich besetzte Matrix. Es ist eine Konsequenz des Geburtstags-Paradoxons, das Übereinstimmungen zwischen zwei *großen Primzahlen* so häufig vorkommen, dass sich das Verfahren lohnt: mit einer großen Primzahl wird die Laufzeit quasi halbiert. Zwei große Primzahlen halbieren wiederum die Laufzeit des Siebs. Nach Aussagen von A. K. Lenstra setzt sich dieses Verhalten auch für eine dritte große Primzahl fort.

Somit können große Primzahlen als ein "günstiger" Weg gesehen werden, die Faktorbasis zu vergrösern. Günstig, weil mit ihnen nicht gesiebt wird. Wir können allerdings nicht zu viele große Primzahlen zulassen. Ein Grund dafür ist, dass die Spärlichkeit der Matrix im lineare-Algebra-Schritt sehr stark abnehemen würde. Der andere Grund ist, dass das Nachhalten der Ergebnisse sehr komplex wird.

Small prime variation

Zum Sammeln von Relationen wärend des Sieb-Prozesses (Algorithmus 2) brauchen kleine Primzahlen und Primzahlpotenzen sehr lange, um bearbeitet zu werden, da ungefähr $1/p$ Zahlen durch p teilbar sind. Zusätzlich gilt, dass der Beitrag, den sie zu den Logarithmen leisten, sehr klein ist. Deswegen sieben wir nicht mit Primzahlpotenzen unter einer sinnvollen Schranke, z.B. 100. Das macht es notwendig, Zahlen zu speichern, deren Restlogarithmus weiter von Null entfernt ist, als der Druchschnitt. Die Erfahrung zeigt allerdings, dass dieses Vorgehen keinen nennenswerten Effekt hat: es ist wichtig, keine glatten Zahlen zu ignorieren. Das kann zur Folge haben, dass einige nicht glatte Zahlen zusätzlich probiert und verworfen werden müssen. Allerdings kann es als Bonus indirekt auf der Suche nach großen Primzahlen behilflich sein.

2.6 Bestimmen von (Quadrat-)Wurzeln in $\mathbb{Z}_p$

Für das Quadratische Sieb braucht man eine Methode um effizient Quadratwurzeln in $\mathbb{Z}_p$ zu bestimmen. Eine solche Methode soll in diesem Abschnitt erläutert werden.

Sei p prim und $a^{\frac{p-1}{2}} = 1 \mod p$. Das bedeutet, $\exists c \in \mathbb{Z}_p$ so dass $a = c^2 \mod p$. Des Weiteren sei n ein quadratischer Nicht-Rest modulo p, d.h. $\left(\frac{n}{p}\right) = -1$. Außerdem sei

$$p - 1 = 2^\alpha \cdot s$$

mit s ungerade.

Man sucht eine Lösung der Gleichung

$$x^2 = a \mod p$$

Um diese zu bestimmen, wählt man zwei Werte

$$b := n^s \mod p \qquad \text{und} \qquad r := a^{\frac{s+1}{2}} \mod p$$

Im Folgenden wird zunächst betrachtet, warum diese Wahl Sinn macht und wie man damit den gesuchten Wert x ermitteln kann.

Für einen Erzeuger g von $\mathbb{Z}_p^*$ gibt es ein β, so dass $g^\beta = n \mod p$ gilt. Der Wert β muss ungerade sein, da $\left(\frac{n}{p}\right) = -1$, also n kein Quadrat modulo p ist. Für die Ordnung von n gilt damit

$$\operatorname{ord}(n) = \operatorname{ord}(g^\beta) = \frac{\operatorname{ord}(g)}{\operatorname{ggT}(\operatorname{ord}(g), \beta)} = \frac{p-1}{v} = \frac{2^\alpha \cdot s}{v}$$

wobei v ungerade ist, da β ungerade ist. Damit ergibt sich $\operatorname{ord}(b) = 2^\alpha$, da

$$\operatorname{ord}(b) = \operatorname{ord}(n^s) = \frac{\operatorname{ord}(n)}{\operatorname{ggT}(\operatorname{ord}(n), s)} = \frac{\frac{2^\alpha \cdot s}{v}}{\frac{s}{v}} = \frac{2^\alpha s v}{sv} = 2^\alpha$$

Das bedeutet, dass b eine primitive 2^α-Einheitswurzel ist, d.h., $b^{2^\alpha} = 1$ aber $b^{2^{\alpha-1}} = -1$.

Betrachtet man r, so fällt auf, dass r "fast" ein Quadrat von a ist:[†]

$$\left(\frac{r^2}{a}\right)^{2^{\alpha-1}} = \left(\frac{a^{s+1}}{a}\right)^{2^{\alpha-1}} = a^{s \cdot 2^{\alpha-1}} = a^{\frac{p-1}{2}} = (c^2)^{\frac{p-1}{2}} = c^{p-1} = 1 \mod p$$

Daraus folgt, dass $\frac{r^2}{a}$ eine $2^{\alpha-1}$-Einheitswurzel ist. Man muss r nun mit Hilfe einer 2^α-Einheitswurzel modifizieren, um ein x zu erhalten, so dass $\frac{x^2}{a} = 1$ gilt. Da b eine primitive 2^α-Einheitswurzel ist, muss somit r mit einer geeigneten Potenz von b multipliziert werden um x zu erhalten.

Mit $b = n^s$ gilt also $(rb^j) = x$ für ein $j \in \{0, \ldots, 2^{\alpha-1}\}$. Schreibt man j in Binärdarstellung, gilt

$$j = j_0 + j_1 2 + j_2 2^2 + \ldots + j_{\alpha-2} 2^{\alpha-2}$$

Im folgenden werden die j_i nacheinander bestimmt.

[†]Wenn r ein Quadrat von a wäre, dann würde gelten: $r^2 = a \Rightarrow \frac{r^2}{a} = 1$.

1. Zur Berechnung von j_0 betrachtet man
$$k = \left(\frac{r^2}{a}\right)^{2^{\alpha-2}} = \pm 1$$
Man setzt
$$j_0 = \begin{cases} 0 & \text{wenn } k = 1 \\ 1 & \text{wenn } k = -1 \end{cases}$$
Damit folgt, dass $\left(\frac{(b^{j_0}r)^2}{a}\right)^{2^{\alpha-2}} = 1$, d.h. $\frac{(b^{j_0}r)^2}{a}$ ist eine $2^{\alpha-2}$-Einheitswurzel, denn für $j_0 = 1$ (also $k = -1$) gilt
$$\left(\frac{(b^{j_0}r)^2}{a}\right)^{2^{\alpha-2}} = b^{2^{\alpha-1}} \cdot \left(\frac{r^2}{a}\right)^{2^{\alpha-2}} = b^{2^{\alpha-1}} \cdot k = (-1)(-1) = 1$$
da $b^{2^{\alpha-1}} = -1$ weil b eine primitive 2^α-Einheitswurzel ist. Für $b_0 = 0$ (also $k = 1$) ist die Betrachtung trivial.

2. Seien j_0 bis j_{l-1} bereits bestimmt, so dass
$$d = \frac{\left(b^{j_0+j_1 2+\ldots+j_{l-1}2^{l-1}} \cdot r\right)^2}{a}$$
eine $2^{\alpha-l-1}$-Einheitswurzel ist. Man setzt
$$j_l = \begin{cases} 0 & \text{wenn } d^{2^{\alpha-l-2}} = 1 \\ 1 & \text{wenn } d^{2^{\alpha-l-2}} = -1 \end{cases}$$
Dies setzt man so lange fort, bis $l = \alpha - 2$, d.h., $d = 1$ (oder eine 2^0-Einheitswurzel) ist. Damit gilt dann
$$\frac{\left(b^j r\right)^2}{a} = 1 \qquad \text{und damit} \qquad \left(b^j r\right)^2 = a = x^2$$

2.9 Beispiel. *Seien* $a = 186$, $n = 3$ *und* $p = 401$. *Dann ergeben sich*
$a^{-1} \bmod p = 235$,
$p - 1 = 400 = 2^4 \cdot 25$, *d.h.,* $s = 25$,
$b = n^s \bmod p = 3^{25} = 268$ *und*
$r = a^{\frac{s+1}{2}} \bmod p = 186^{13} = 103$

j_0: $k = (r^2 \cdot a^{-1})^{2^{\alpha-2}} \bmod p = (103^2 \cdot 235)^{2^2} = 98^4 = -1 \qquad \Rightarrow j_0 = 1$

j_1: $d = \left((b^{j_0}r)^2 \cdot a^{-1}\right)^{2^{\alpha-1-2}} \bmod p = \left((268 \cdot 103)^2 \cdot 235\right)^2 = 1 \qquad \Rightarrow j_1 = 0$

j_2: $d = \left((b^{j_0+2j_1}r)^2 \cdot a^{-1}\right)^{2^{\alpha-2-2}} \bmod p = (268 \cdot 103)^2 \cdot 235 = -1 \qquad \Rightarrow j_2 = 1$

$\Rightarrow \quad j = j_0 + 2j_1 + 2^2 j_2 = 1 + 2^2 \cdot 1 = 5$
$\Rightarrow \quad x = b^j r \bmod p = 268^5 \cdot 103 = 304$
Test: $304^2 = 186 \bmod 401$

2.7 Weitere Faktorisierungsalgorithmen

Es gibt noch einige weitere Algorithmen zum Faktorisieren von großen Zahlen, auf die wir hier nicht detailiert eingehen wollen.

1. Faktorisieren mit Elliptischen Kurven
   - Verallgemeinerung der $(p-1)$-Methode
   - gut geeignet, wenn der RSA Modul einen kleinen Primteiler hat
2. General Number Field Sieve (GNFS)
   - Verbesserung des Quadratischen Siebs
   - im Allgemeinen beste Möglichkeit um RSA Moduli zu faktorisieren

Die größte RSA-Challenge, die bisher faktorisiert werden konnte (BSI mittels GNFS, Bonn), hatte 640 Bits (RSA-640).

2.8 Konsequenzen für RSA-Schlüsselerzeugung

Als Konsequenz der vorhergehenden Kapitel zum Brechen von RSA und zur Faktorisierung empfiehlt es sich, bei der Erzeugung von RSA-Schlüsseln folgendermaßen vorzugehen:

1. wähle Primzahlen p, q zufällig aber etwa mit gleicher Bitlänge:
$$p, q \in \{2^b, \ldots, 2^{b+1}\} \qquad p, q \text{ prim}$$
2. wähle öffentlichen Exponenten $e \geqslant 17$
3. teste, ob $\mathrm{ggT}\,(e, (p-1)(q-1)) = 1$
4. wenn nicht, wähle neues p und q

Wenn man e neu wählt, würde man einem Angreifer einen Teiler von $(p-1)(q-1)$ verraten. Man weiß nicht, ob dies ein unmittelbares Problem darstellt, jedoch empfiehlt es sich, jegliche unnötige Preisgabe von Informationen zu vermeiden.

Kapitel 3

Diskrete Logarithmen

Sei G eine endliche, abelsche Gruppe mit $g \in G$ und $a \in \langle g \rangle$, d.h. es existiert ein $x \in \mathbb{N}$, so dass

$$a = g^x$$

Man bezeichnet x als den diskreten Logarithmus (DL) von a zur Basis g und schreibt

$$x = \log_g a.$$

DL-Problem: Das DL-Problem beschreibt die Suche nach dem Exponenten x bei gegebenem a und g. Es gibt Gruppen, in denen Das DL-Problem leicht zu lösen ist:

- $(\mathbb{Z}_2^*, \cdot), (\mathbb{Z}_{17}^*, \cdot)$, da diese Gruppen sehr klein sind
- $(\mathbb{Z}_p, +)$ mit p prim: Man bestimmt x aus der Gleichung $a = x \cdot g$ mit gegebenem a und g das x, indem man mit der Inversen g^{-1} multipliziert.

Es gibt jedoch auch Gruppen, wo man hofft, dass es keine effizienten Algorithmen zum Berechnen des diskreten Logarithmus gibt:

- $(\mathbb{Z}_p^*, \cdot)$
- $(\mathbb{F}_q^*, \cdot)$ mit $q = p^n$ und p prim
- Elliptische Kurven

Anwendungen: In folgenden Anwendungen ist der diskrete Logarithmus von großer Bedeutung:

- Diffie-Hellman-Schlüsselaustausch
- El Gamal
- DSA (DSS)

wobei der Diffie-Hellmann-Schlüsselaustausch auf folgendem Diffie-Hellman-Problem beruht: Bei gegebenen Werten g, g^a und g^b ist es schwierig, g^{ab} zu berechnen.
Es ist eine bis jetzt offene Frage, ob das Diffie-Hellman-Problem und das DL-Problem gleich schwierig sind. Klar ist, dass wer immer das DL-Problem lösen kann auch das DH-Problem lösen kann. Die Umkehrung konnte bis jetzt weder gezeigt noch widerlegt werden.

3.1 Algorithmen um den diskreten Logarithmus zu berechnen

Im Folgenden werden einige Algorithmen vorgestellt, die den diskreten Logarithmus mehr oder weniger effizient berechnen können. Für alle diese Algorithmen gilt:
Gegeben sei $g, a \in \langle g \rangle = G$. Gesucht wird $x = \log_g a$.

3.1.1 Brute-Force

Man testet für alle $x \in \{0, \ldots, \mathrm{ord}(g) - 1\}$ ob $g^x = a$.

Aufwand: Der Aufwand für diese Methode ist $O(\mathrm{ord}(g))$ bzw. $O(|G|)$ falls g ein Erzeuger der Gruppe G ist.

Bemerkungen: Dieser Algorithmus funktioniert für jede Gruppe G.

3.1.2 Baby-Step, Giant-Step

Die Idee ist, dass sich jede Zahl $x \in \{0, \ldots, |G|\}$ schreiben lässt als

$$x = x_0 + x_1 \lfloor \sqrt{|G|} \rfloor$$

für $x_0, x_1 \in \{0, \ldots, \lfloor \sqrt{|G|} \rfloor\}$. Der Algorithmus nutzt diese einfache Erkenntnis um den Aufwand der Berechnung erheblich zu verbessern.

Man berechnet für alle $x_0, x_1 \in \{0, \ldots, \lfloor \sqrt{|G|} \rfloor\}$

$$\frac{g^{x_0}}{a} \qquad \text{und} \qquad g^{-\lfloor \sqrt{|G|} \rfloor \cdot x_1}$$

und speichert diese Werte in einer **sortierten** Tabelle. Wird in den Tabellen eine Kollision gefunden, d.h. es gibt Werte x_0, x_1, so dass die jeweiligen Werte in der zweiten Spalte identisch sind, dann gilt:

$$\frac{g^{x_0}}{a} = g^{-\lfloor \sqrt{|G|} \rfloor \cdot x_1}$$

das heist

$$a = g^{x_0 + \lfloor \sqrt{|G|} \rfloor \cdot x_1}$$

Daraus folgt:

$$x = x_0 + \lfloor \sqrt{|G|} \rfloor \cdot x_1 \bmod |G|$$

Aufwand: Der Aufwand für diesen Algorithmus ist $O(\sqrt{|G|})$, also sehr viel geringer als obiger Brute-Force-Ansatz.

Bemerkungen: Auch dieser Algorithmus funktioniert für jede Gruppe G, da nur Gruppenoperationen verwendet werden. Insbesondere sollten für Kryptographieverfahren, die auf dem diskreten Logarithmus basieren, nur Gruppen mit $|G| \geqslant 2^{160}$ genutzt werden um den Aufwand groß genug zu halten. Wie wir im Abschnitt 5 sehen werden, kann man beweisen, dass es keine effizienteren Algorithmen zum Berechnen von diskreten Logarithmen gibt, die nur die Gruppenoperationen verwenden.

3.1.3 Silver-Pohlig-Hellman-Algorithmus

Sei $|G| = n$ und $n = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r}$. Der Silver-Pohlig-Hellman-Algorithmus berechnet den diskreten Logarithmus in zwei Schritten.

- Bestimme den Logarithmus $x \bmod p_i^{\alpha_i}$ mit $i \in \{1, \ldots, r\}$
- Berechne mit Hilfe des Chinesischen Restsatzes $x \bmod n$

Der zweite Schritt ist einfach und effizient und soll an dieser Stelle nicht näher betrachtet werden. Im Folgenden wird der erste Schritt näher erläutert:

Algorithmus zum Bestimmen von $x \bmod p_i^{\alpha_i}$

- Berechne $r_{p_i,j} = (g^{\frac{n}{p_i}})^j \bmod G$ für alle $j \in \{0, \ldots, p_i - 1\}$ und speichere diesen Wert in einer Tabelle. Diese Vorbereitung ist unabhängig von a, d.h. diese Tabelle muß einmal bestimmt werden und kann für jedes $a \in G$ benutzt werden.
- Berechne $a^{\frac{n}{p_i}}$ und vergleiche mit den Elementen aus der Tabelle. Wähle j_0, so dass $a^{\frac{n}{p_i}} = r_{p_i,j_0}$.
- Setze $a_1 = \frac{a}{g^{j_0}}$.
- Berechne $a_1^{\frac{n}{p_i^2}}$ und vergleiche mit den Elementen aus der Tabelle. Wähle j_1, so dass $a_1^{\frac{n}{p_i^2}} = r_{p_i,j_1}$.
- Für $k \in \{2, \ldots, \alpha - 1\}$ setze $c_k = \sum_{l=0}^{k} j_l p_i^l$ und $a_k = \frac{a_{k-1}}{g^{c_k}}$ und berechne $a_k^{\frac{n}{p_i^{k+1}}}$ und vergleiche mit den Elementen aus der Tabelle. Wähle j_k, so dass $a_k^{\frac{n}{p_i^{k+1}}} = r_{p_i,j_k}$.

Dann gilt $x = j_0 + j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1} \bmod p_i^{\alpha}$.

Warum funktioniert dieser Algorithmus? Sei

$$x = j_0 + j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1} \bmod p_i^{\alpha}$$

Dann gilt:

$$\begin{aligned}
a^{\frac{n}{p_i}} &= (g^x)^{\frac{n}{p_i}} \\
&= (g^{j_0 + j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1}})^{\frac{n}{p_i}} \\
&= (g^{j_0})^{\frac{n}{p_i}} \cdot (g^{j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1}})^{\frac{n}{p_i}} \\
&= r_{p_i,j_0} \cdot (g^{p_i(j_1 + j_2 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-2})})^{\frac{n}{p_i}} \\
&= r_{p_i,j_0} \cdot (g^{j_1 + j_2 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-2}})^{n} \\
&= r_{p_i,j_0}
\end{aligned}$$

□

Weiterhin gilt für $a_1 = \frac{a}{g^{j_0}}$:

$$\begin{aligned}
a_1^{\frac{n}{p_i^2}} &= \frac{a^{\frac{n}{p_i^2}}}{(g^{j_0})^{\frac{n}{p_i^2}}} \\
&= \left(\frac{g^{j_0 + j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1}}}{g^{j_0}} \right)^{\frac{n}{p_i^2}} \\
&= (g^{j_1 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-1}})^{\frac{n}{p_i^2}} \\
&= (g^{j_1 p_i})^{\frac{n}{p_i^2}} \cdot (g^{p_i^2(j_2 + j_3 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-3})})^{\frac{n}{p_i^2}} \\
&= r_{p_i,j_1} \cdot (g^{j_2 + j_3 p_i + \ldots + j_{\alpha-1} p_i^{\alpha-3}})^{n} \\
&= r_{p_i,j_1}
\end{aligned}$$

□

Die restlichen j_k folgen per Induktion.

Aufwand: Es müssen p_i Werte für die Tabelle berechnet werden. Zum Berechnen von j_k benötigt man $O(\alpha_i)$ Operationen. Daraus folgt, dass dieser Algorithmus dann praktisch durchführbar ist, wenn die Gruppenordnung n nur von relativ kleinen Primzahlen (z.B. $n = 2^{1024}$) geteilt wird.

Bemerkungen: Dieser Algorithmus funktioniert unabhängig von der Art der Gruppe, d.h. sowohl für $\mathbb{F}_q^*$ als auch für Elliptische Kurven.

3.1.4 Index Calculus

Dieser Algorithmus funktioniert im Gegensatz zu den oberen nur für die multiplikative Gruppe von endlichen Körpern mit kleiner Charakteristik und großem Erweiterungsgrad,

d.h. für $\mathbb{F}_q^*$ mit $q = p^n$, wobei p klein. Es ist nicht bekannt, wie man diesen Algorithmus z.B. für Elliptische Kurven anpassen kann. Deshalb kann man den Sicherheitsparameter in Kryptosystemen, die auf elliptischen Kurven basieren, deutlich geringer wählen.

Es gelte

$$\mathbb{F}_{p^n} \cong \mathbb{F}_p[x]/(f),$$

wobei $f \in \mathbb{F}_p[x], \deg(f) = n$ und f irreduzibel. D.h. zu einem Element $a \in \mathbb{F}_{p^n}$ gehört ein Polynom $a(x)$ mit $grad(a(x)) < n$. Für Elemente $c(x) \in \mathbb{F}_{p^n}$ mit $c \in \mathbb{F}_p$ kann der diskrete Logarithmus bezüglich eines Generators $g \in \mathbb{F}_{p^n}$ leicht berechnet werden, da p nach Voraussetzung klein ist. Sei $\langle g \rangle = \mathbb{F}_{p^n}^*$, dann ist

$$g^{\frac{p^n-1}{p-1}} \in \mathbb{F}_p^*$$

denn

$$\left(g^{\frac{p^n-1}{p-1}}\right)^{p-1} = 1$$

und

$$\langle g^{\frac{p^n-1}{p-1}} \rangle = \mathbb{F}_p^*$$

D.h. um die diskreten Logarithmen von Elementen $c \in \mathbb{F}_p$ zu erhalten, kann man eine Tabelle mit allen Elementen

$$\left(g^{\frac{p^n-1}{p-1}}\right)^j \qquad \text{mit} \qquad j \in \{0, \ldots, p-2\}$$

berechnen.

Funktionsweise des Algorithmus Wähle eine Faktorbasis $B \subseteq \mathbb{F}_p[x]$. Zum Beispiel $B = \{a \in \mathbb{F}_p[x] | grad(a) \leqslant k, a \text{ irreduzibel}\}$ für eine geeignete Konstante k. Diese Elemente werden im weiteren als Elemente in $\mathbb{F}_{p^n}$ aufgefasst.

1. Stufe: In der ersten Stufe sollen die diskreten Logarithmen aller Elemente von B bestimmt werden.

- Wähle $t \in_R \{1, \ldots, p^n - 2\}$.
- Berechne $c = g^t$, also $c(x) = g(x)^t \bmod f(x)$.
- Teste, ob das Element $c(x)$ ein Produkt von Elementen $b(x) \in B$ ist, also über der Faktorbasis B zerfällt.

War der obige Test erfolgreich, gibt es Exponenten $\alpha_b \in \mathbb{N}$ und eine Konstante $c_0 \in \mathbb{F}_p$, so dass

$$c(x) = \left(\sum_{b(x) \in B} b(x)^{\alpha_b} \right) \cdot c_0$$

Dann gilt:

$$\log_g c(x) \;=\; \log_g \left[\left(\sum_{b(x)\in B} b(x)^{\alpha_b} \right) \cdot c_0 \right]$$

und damit

$$t \;=\; \log_g c_0 + \sum_{b(x)\in B} \left(\alpha_b \log_g b(x) \right)$$

Für jedes $c(x) \in \frac{\mathbb{F}_p[x]}{(f)}$, dass über der Faktorbasis B zerfällt, erhält man eine lineare Gleichung in den gesuchten Werten $\log_g b(x)$. Für genügend viele Elemente $c(x)$, die über B zerfallen, können diese Werte daher durch Lösen eines linearen Gleichungssystems berechnet werden.

2. Stufe: In der zweiten Stufe wird $\log_g a$ für ein gegebenes Element $a \in \mathbb{F}_{p^n}^*$ berechnet.

- Wähle $t \in_R \{1, \ldots, p^n - 2\}$.
- Berechne $y = a \cdot g^t$, also $y(x) = a(x) \cdot g(x)^t \bmod f$.
- Teste, ob $y(x)$ über der Faktorbasis B zerfällt.
- Wenn nicht, wähle neues $t \in_R \{1, \ldots, p^n - 2\}$.

War der obige Test erfolgreich, gibt es Exponenten $\alpha'_b \in \mathbb{N}$ und eine Konstante $y_0 \in \mathbb{F}_p$, so dass

$$y(x) = \left(\sum_{b(x)\in B} b(x)^{\alpha'_b} \right) \cdot y_0$$

Es gilt:

$$\begin{aligned} \log_g y(x) &= \log_g (a \cdot g^t) \\ &= \log_g a + \log_g g^t \\ &= \log_g a + t \end{aligned}$$

und daher

$$\begin{aligned} \log_g a &= \log_g \left[\left(\sum_{b(x)\in B} b(x)^{\alpha'_b} \right) \cdot y_0 \right] - t \\ &= \sum_{b(x)\in B} \alpha'_b \log_g b(x) - t + \log_g y_0 \end{aligned}$$

Da die Werte $\log_g b(x)$ nach der ersten Stufe bekannt sind, ist somit $\log_g a$ berechnet.

Kapitel 4

Einführung in die kurvenbasierte Kryptographie

4.1 Kurven

4.1.1 Affine Kurven

Wir fangen mit einer „low brow“ Einführung in die Theorie der ebenen Kurven im affinen und im projektiven Raum an.

4.1 Definition. (Affine ebene Kurve) *Sei $\mathbb{F}$ ein Körper und $f \neq 0$ ein Polynom in den zwei Variablen x und y, mit Koeffizienten aus $\mathbb{F}$ von denen nur endlich viele ungleich Null sind. Dann nennen wir die Menge der Nullstellen*

$$C_f(\mathbb{F}) = \{(a, b) \in \mathbb{F} \times \mathbb{F} \ : f(a, b) = 0\}$$

eine affine ebene Kurve.

Der *zweidimensionale affine Raum* über $\mathbb{F}$ ($\mathbb{F} \times \mathbb{F}$) wird im folgenden auch $\mathrm{A}^2(\mathbb{F})$ bezeichnet.

4.2 Definition. (Singularität) *Eine affine Kurve heisst* singulär *in einem Punkt $(a, b) \in C_f(\mathbb{F})$, falls beide Ableitungen in dem Punkt verschwinden, d.h.*

$$f(a, b) = 0 \ , \ \frac{\partial f}{\partial x}(a, b) = 0 \ \textit{und} \ \frac{\partial f}{\partial y}(a, b) = 0 \ .$$

Eine Kurve heisst nichtsingulär*, wenn im algebraischen Abschluss $\bar{\mathbb{F}}$ von $\mathbb{F}$ kein Punkt $(a, b) \in \mathrm{A}^2(\bar{\mathbb{F}})$ existiert für den beide Ableitungen verschwinden.*

Warum die Nichtsingularität eine wichtige Eigenschaft ist werden wir bei der Einführung des *geometrischen Gruppengesetzes* in Abschnitt 4.2.2 sehen.

4.3 Beispiel. *Sei p eine Primzahl. Betrachen wir die Kurve welche durch folgendes Polynom über dem endlichen Körper $\mathbb{F}_p$ definiert ist:*

$$f(x, y) = y^2 - x^3 - x$$

Die Ableitungen von f sind:

$$\frac{\partial f}{\partial x}(x, y) = -3x^2 - 1 \text{ und } \frac{\partial f}{\partial y}(x, y) = 2y$$

Für einen singulären Punkt $(a, b) \in \bar{\mathbb{F}}_p \times \bar{\mathbb{F}}_p$ müsste also gelten

$$b^2 = a^3 + a \text{ und } -3a^2 - 1 = 0 \text{ und } 2b = 0$$

Wenn $p \neq 2$ dann führt dies zu einem Widerspruch. Das bedeutet: Für $p > 2$ gibt es keine singulären Punkte auf $C_f(\bar{\mathbb{F}}_p)$, das bedeutet die Kurve $C_f(\mathbb{F}_p)$ ist nichtsingulär.

4.1.2 Projektive Kurven

Betrachten wir weiterhin die Kurve $f(x, y) = y^2 - x^3 - x$ aus obigem Beispiel. Die Nullstellen dieser Gleichung entsprechen der Menge aller Lösungen der Gleichung

$$y^2 = x^3 + x \tag{4.1}$$

Sei $(a, b) \in \mathrm{A}^2(\mathbb{F})$ eine solche Lösung. Sei $c \in \mathbb{N}$ eine beliebige Zahl ungleich Null. Wenn wir $a' = ac$ und $b' = bc$ definieren, so gilt:

$$\left(\frac{b'}{c}\right)^2 = \left(\frac{a'}{c}\right)^3 + \frac{a'}{c}$$

Dies können wir mit c^3 multiplizieren. Dann erhalten wir $b'^2 c = a'^3 + a' c^2$. Das Tripel (a', b', c) ist also eine Lösung der Gleichung

$$Y^2 Z = X^3 + XZ^2 \tag{4.2}$$

Warum aber sollte man von der leichteren Gleichung (4.1) zur komplizierteren Gleichung (4.2) übergehen? Der Grund ist, dass (4.2) noch weitere Lösungen hat. Welche sind das? Wir nehmen an, dass das Tripel (a, b, c) eine Lösung von (4.2) sei. Das heisst:

$$b^2 c = a^3 + ac^2$$

Dann gibt es zwei Möglichkeiten:

1. Wenn $c \neq 0$, dann können wir durch c^3 teilen. Dann ist $\frac{a}{c}, \frac{b}{c}$ eine Lösung von (4.1).
2. Wenn $c = 0$, dann ist $a^3 = 0$, und damit $a = 0$. b kann beliebig sein. In diesem Falle gibt es keine entsprechende Lösung von (4.1).

Ausserdem sehen wir, dass wenn (a, b, c) eine Lösung von (4.2) ist, dann auch $(at, bt, ct), t \neq 0$. Es gilt $(\frac{a}{c}, \frac{b}{c}) = (\frac{at}{ct}, \frac{bt}{ct})$.

Diese Überlegungen motivieren die folgende Definition:

4.4 Definition. (Zweidimensionaler projektiver Raum $\mathbb{P}^2$)

1. *Wir nennen zwei Punkte (a, b, c) und $(a', b', c') \in \mathbb{F}^3$ äquivalent, falls es ein $t \in \mathbb{F}\{0\}$ gibt mit*
$$a = ta' \ , \ b = tb' \ , \ c = tc'$$
Wir bezeichnen diese Äquivalenzrelation mit $\sim$.

2. *Den zweidimensionale projektive Raum $\mathbb{P}^2$ definieren wir als den Quotienten von $(\mathbb{F} \times \mathbb{F} \times \mathbb{F} \setminus \{(0, 0, 0)\})$ nach der Äquivalenzrelation $\sim$:*
$$\mathbb{P}^2 := (\mathbb{F} \times \mathbb{F} \times \mathbb{F} \setminus \{(0, 0, 0)\})/ \sim$$

Der projektive Raum $\mathbb{P}^2(\mathbb{F})$ ist also die Menge der Äquivalenzklassen von $\sim$. Wir bezeichnen die Äquivalenzklasse eines Punktes (a, b, c) mit $[a : b : c]$, und es gilt

$$[a : b : c] = [a' : b' : c'] \Leftrightarrow \exists\ t \neq 0 \in \mathbb{F} \text{ mit } a = ta' \ , \ b = tb' \ , \ c = tc'$$

4.5 Definition. *Ein Polynom $g \in \mathbb{F}[X, Y, Z]$ vom Grad d heisst homogen, falls*

$$g(ta, tb, tc) = t^d g(a, b, c) \ .$$

Es ist leicht zu zeigen, dass ein Polynom ist homogen vom Grad d genau dann wenn es Summe von Monomen vom Grad d ist.

Das bedeutet also: Bezüglich der Nullstellen von g ist es irrelevant wie man einen Punkt aus $[a : b : c]$ schreibt (also: welchen Vertreter der Äquivalenzklasse man wählt). Wir können nun Kurven im projektiven Raum definieren, analog zu den affinen Kurven:

4.6 Definition. (Projektive Kurve) *Sei g ein homogenes Polynom in $\mathbb{F}[X, Y, Z]$. Wir bezeichnen die Menge $C_g(\mathbb{F})$ der Nullstellen von g in $\mathbb{P}^2(\mathbb{F})$, d.h.*

$$C_g(\mathbb{F}) = \{[a : b : c] \in \mathbb{P}^2(\mathbb{F}) : g(a, b, c) = 0\}$$

als projektive ebene Kurve *über $\mathbb{F}$.*

Eine projektive Kurve ist also die Menge der Nullstellen des Polynoms g. Dazu nun ein Beispiel:

Betrachten wir weiterhin unser Beispiel

$$f(x, y) = y^2 - x^3 - x$$

$C_f(\mathbb{F})$ ist also die Menge der Lösungen der Gleichung (4.1). Sei

$$g(X, Y, Z) = Y^2 Z - X^3 - XZ^2$$

Dann ist $C_g(\mathbb{F})$ die Menge der Lösungen der Gleichung (4.2): sie enthält (durch die Einbettung i) sowohl die affine Lösungsmenge $C_f(\mathbb{F})$, als auch der „unendlich ferne“ Punkt ∞.

Für alle $(a, b) \in C_f(\mathbb{F})$ liegt $[a : b : 1]$ in $C_g(\mathbb{F})$. Ausserdem haben wir gesehen dass $C_g(\mathbb{F})$ noch einen weiteren Punkt enthält, nämlich $[0 : 1 : 0]$. Das heisst es gilt

$$C_g(\mathbb{F}) = C_f(\mathbb{F}) \cup \{[0 : 1 : 0]\} \ . \tag{4.3}$$

Wir haben also die affine Kurve in eine entsprechende projektive Kurve eingebettet, die darüber hinaus noch einen weiteren Punkt enthält. Dieser Punkt wird auch der *Punkt im Unendlichen* genannt, und mit ∞ (oder auch 0) bezeichnet.

4.7 Satz (Homogenisierung)**.** *Sei $f \neq 0$ ein beliebiges Polynom in $\mathbb{F}[x, y]$, also:*

$$f(x, y) = \sum_{\nu_1, \nu_2 \geqslant 0} \gamma_{\nu_1, \nu_2} x^{\nu_1} y^{\nu_2}$$

Sei $d := \deg(f) = \max\{\nu_1 + \nu_2 : \gamma_{\nu_1, \nu_2} \neq 0\}$ der totale *Grad von f. Dann gilt für das Polynom*

$$g(X, Y, Z) := Z^d f(X/Z, Y/Z)$$

die Gleichheit

$$g(X, Y, Z) = \sum_{\nu_1, \nu_2 \geqslant 0, \nu_1 + \nu_2 \leqslant d} \gamma_{\nu_1, \nu_2} X^{\nu_1} Y^{\nu_2} Z^{d - \nu_1 - \nu_2} \ ,$$

g is homogen vom Grad d und erfüllt $g(a, b, 1) = f(a, b) \forall (a, b) \in \mathrm{A}^2(\mathbb{F})$. Ausserdem lässt sich f zu g abbilden durch die Abbildung

$$\begin{aligned} i \ &: \mathrm{A}^2(\mathbb{F}) \to \mathbb{P}^2(\mathbb{F}) \\ &(a, b) \mapsto [a : b : 1] \ . \end{aligned}$$

Wenn sich ein Punkt $[a : b : c] \in C_g(\mathbb{F})$ als $i(x)$ für ein $x \in \mathrm{A}^2$ schreiben lässt, so liegt x in $C_f(\mathbb{F})$.

Proof. Man sieht sofort dass das Polynom homogen vom Grad d ist, und dass

$$g(a, b, 1) = f(a, b) \tag{4.4}$$

Daraus folgt, dass

$$i((a, b)) = [a : b : 1] \in C_g(\mathbb{F}) \ \ \forall \ \ (a, b) \in C_f(\mathbb{F})$$

Wenn umgekehrt ein Punkt $[a : b : 1]$ in $C_g(\mathbb{F})$ liegt, dann gilt $g(a, b, 1) = 0$. Nach]eqref-eq:gab1:eq:fab ist das genau dann der Fall, wenn $f(a, b) = 0$. □

Eine andere Schreibweise dafür ist

$$C_g(\mathbb{F}) \cap \mathrm{A}^2(\mathbb{F}) = C_f(\mathbb{F})$$

(vgl. (4.3)).

Die Nichtsingularität definieren wir im projektiven Raum analog zum affinen Raum.

4.8 Definition. *Sei g ein homogenes Polynom in $\mathbb{F}[X,Y,Z]$ vom grad d.*

1. *Die projektive ebene Kurve $C_g(\mathbb{F})$ heisst* singulär *im Punkt $P=[a:b:c]\in C_g(\mathbb{F})$, falls alle Ableitungen von g in P verschwinden, d.h.*
$$\frac{\partial g}{\partial X}(a,b,c)=\frac{\partial g}{\partial Y}(a,b,c)=\frac{\partial g}{\partial Z}(a,b,c)=0$$
2. *$C_g(\mathbb{F})$ heisst nichtsingulär, falls $C_g(\mathbb{F})$ keinen singulären Punkt enthält.*

4.9 Satz. *Sei $f(x,y)$ ein beliebiges Polynom von totalem Grad d und und $g(X,Y,Z)=Z^dG(X/Z,Y/Z)$ seine Homogenisierung (ebenfalls also vom Grad d). Für jeden Punkt $P=i(Q)\in C_g(\mathbb{F})\backslash\{[0:1:0]\}$ gilt: $C_g(\mathbb{F})$ ist genau dann singulär in P, wenn die affine Kurve $C_f(\mathbb{F})$ singulär in Q ist.*

(Beweisidee: Der Punkt $[0:1:0]$ ist nie singulär, denn es gilt immer $\frac{\partial g}{\partial Z}(0,1,0)=1$. Wenn man also prüfen will ob die Kurve $C_g(\mathbb{F})$ nichtsingulär ist, dann genügt es zu prüfen ob $C_f(\mathbb{F})$ nichtsingulär ist.)

4.2 Elliptische Kurven

4.10 Definition. Elliptische Kurve *Eine **elliptische Kurve** E über dem Körper K ist eine „nichtsinguläre“ Kurve, gegeben durch eine Gleichung der Form*
$$E \quad : \quad y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6 \tag{4.5}$$
wobei a_1, a_2, a_3, a_4 und a_6 Elemente aus K sind. Eine Gleichung obiger Form nennt man Weierstrass-Gleichung.

Mit anderen Worten, die Kurve E ist durch eine Gleichung
$$F(x,y):=y^2+a_1xy+a_3y-(x^3+a_2x^2+a_4x+a_6)$$
gegeben.

4.2.1 Vereinfachte Weierstrass Gleichungen über endlichen Körpern

4.11 Definition. *Zwei elliptische Kurven E_1,E_2 über $\mathbb{F}$, gegeben als Weierstrass Gleichungen*
$$\begin{aligned} E_1 : y^2+a_1xy+a_3y &= x^3+a_2x^2+a_4x+a_6\\ E_2 : y^2+\bar{a}_1xy+\bar{a}_3y &= x^3+\bar{a}_2x^2+\bar{a}_4x+\bar{a}_6 \end{aligned}$$
werden als isomorph über $\mathbb{F}$ *bezeichnet, falls $u,r,s,t\in\mathbb{F}$ existieren, sodass durch eine Veränderung der Variablen der Form*
$$(x,y)\to(u^2x+r,u^3y+u^2sx+t)$$
die eine Gleichung in die andere umgeformt werden kann. Diese Umformung wird admissible change of variables *genannt.*

4.12 Beispiel. *Falls* $\mathrm{char}(\mathbb{F}) \neq 2, 3$*, dann kann* E *(definiert durch* (4.5)*) durch*

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right)$$

umgeformt werden zu

$$y^2 = x^3 + ax + b \tag{4.6}$$

mit $a, b \in \mathbb{F}$*.*

4.13 Bemerkung. *1. Sei* $L/\mathbb{F}$ *ein Erweiterungkörper. Die Menge der affinen L-rationalen Punkte* $E(L)$ *auf* E *wird dann durch die Menge*

$$\{(x_0, y_0) : y_0^2 + a_1 x_0 y_0 + a_3 y_0 = x_0^3 + a_2 x_0^2 + a_4 x_0 + a_6\} \cup \{\infty\}$$

gegeben.

2. Selbstverständlich kann die Kurve auch als projektive Kurve auffassen werden. Dazu schreiben wir (z.B. in char $\neq 2, 3$*) zunächst das homogene Polynom für die Kurve auf:*

$$y^2 z = x^3 + axz^2 + bz^3$$

wobei $a, b \in \mathbb{F}$*. Zwei Punkte* $[x_0 : y_0 : z_0]$ *und* $[x_1 : y_1 : z_1]$ *sind äquivalent, falls ein* λ *in* $\mathbb{F}$ *existiert mit* $[x_0 : y_0 : z_0] = \lambda[x_1 : y_1 : z_1]$*. Die affinen Punkte erhalten wir, in dem wir* $z_0 \neq 0$ *setzen. Dazu kommt ein weiterer Punkt für* $z_0 = 0$ *(der unendlich ferne Punkte* ∞*). Der projektive Punkt* $[x_0 : y_0 : z_0]$ *ist der affine Punkt* $(x_0/z_0, y_0/z_0)$*. Insbesondere lässt der affine Punkt* (x_0, y_0) *sich als* $[x_0 : y_0 : 1]$ *in projektiven Koordinaten darstellen.*

3. Ein singulärer Punkt einer projektiven Kurven $F(x, y, z) = 0$ *ist nach Theorem 4.9 ein Punkt* $P = [x_0 : y_0 : z_0]$*, der die Kurvengleichung erfüllt und für den zusätzlich*

$$\frac{\partial F}{\partial x}(P) = \frac{\partial F}{\partial y}(P) = \frac{\partial F}{\partial z}(P) = 0$$

gilt. Wir können leicht nachprüfen, dass z.B. im Fall char $\mathbb{F} = 2, 3$ *durch die Bedingung*

$$\Delta = 4a^3 + 27b^2 \neq 0$$

singuläre Punkte ausgeschlossen werden.

Proof. Wenn ein Kurve singulär ist, muss mindestens ein Punkt exestieren, derfolgende Gleichungen erfüllt:

(a) $y^2 = x^3 + ax + b$

(b) $\frac{df}{dx} = 3x^2 + a = 0$

(c) $\frac{df}{dy} = 2y = 0$

Aus Gleichung (b) ergibt sich: $x^2 = \frac{-a}{3}$.

Somit ergibt sich aus Gleichung (a)(c): $y = 0 \Rightarrow 0 = x^3 + ax + b$.

$\Leftrightarrow x(x^2 + a) + b = 0$
$\Leftrightarrow x(\frac{-a}{3} + a) + b = 0$
$\Leftrightarrow x = \frac{-3b}{2a}$
(b)$\Rightarrow 3\frac{9b^2}{4a^2} = 0$
$\Leftrightarrow 4a^3 + 27b^2 = 0$

Somit muss für eine nichtsinguläre Kurve gelten: $4a^3 + 27b^2 \neq 0$

□

Die Zahl

$$\Delta = 4a^3 + 27b^2 \neq 0$$

heisst Diskriminante. *Alternative Formen sind:*

$$\Delta = -12(4a^3 + 27b^2) \qquad \Delta = -16(4a^3 + 27b^2) \qquad \Delta = -1728(4a^3 + 27b^2)$$

Diese Formen sind Äquivalent, da 12, 16 und 1728 nur durch die Primfaktoren 2 und 3 teilbar sind.

Ein singulärer Punkt (x_0, y_0) in einer Körpererweiterung L von $\mathbb{F}$ ist dann ein Punkt in $E(L)$ mit $2y_0 = 0$ und $3x_0^2 + a = 0$.

Eine Vorstellung einer elliptischen Kurve kann Abbildung 4.1 geben. Diese Darstellung hat jedoch einen Nachteil: Der unendliche Punkt lässt sich nicht darstellen. Dafür muss man die Kurve in dem projektiven Raum einbetten.

4.2.2 Geometrisches Gruppengesetz

4.14 Bemerkung. *Die Menge der $\mathbb{F}$-rationalen Punkte vereinigt mit dem unendlich fernen Punkt ∞ auf E bildet eine abelsche Gruppe. Falls $\mathbb{F}$ algebraisch abgeschlossen, trifft eine Gerade die elliptische Kurve immer in drei Punkten (mit Multiplizitäten gezählt). Dies folgt aus einem wichtigen Satz von Bezout. Falls die elliptische Kurve über einem Korper $\mathbb{F}$ definiert ist und zwei Schnittpunkte in einer Körpererweiterung $L/\mathbb{F}$ liegen, dann liegt auch der dritte Schnittpunkt in L.*

Seien $P, Q \in E(\mathbb{F})$.

(a) Falls $P \neq Q$: Verbinde P und Q durch eine Gerade.
Falls $P = Q$, sei die Gerade die Tangente von P an der Kurve.
Sei R der dritte Schnittpunkt der Gerade mit E.

(b) Setze $P+Q :=$ Spiegelung von R an der x-Achse. Falls $P, Q \in E(\mathbb{F})$, dann gilt $P+Q \in E(\mathbb{F})$.

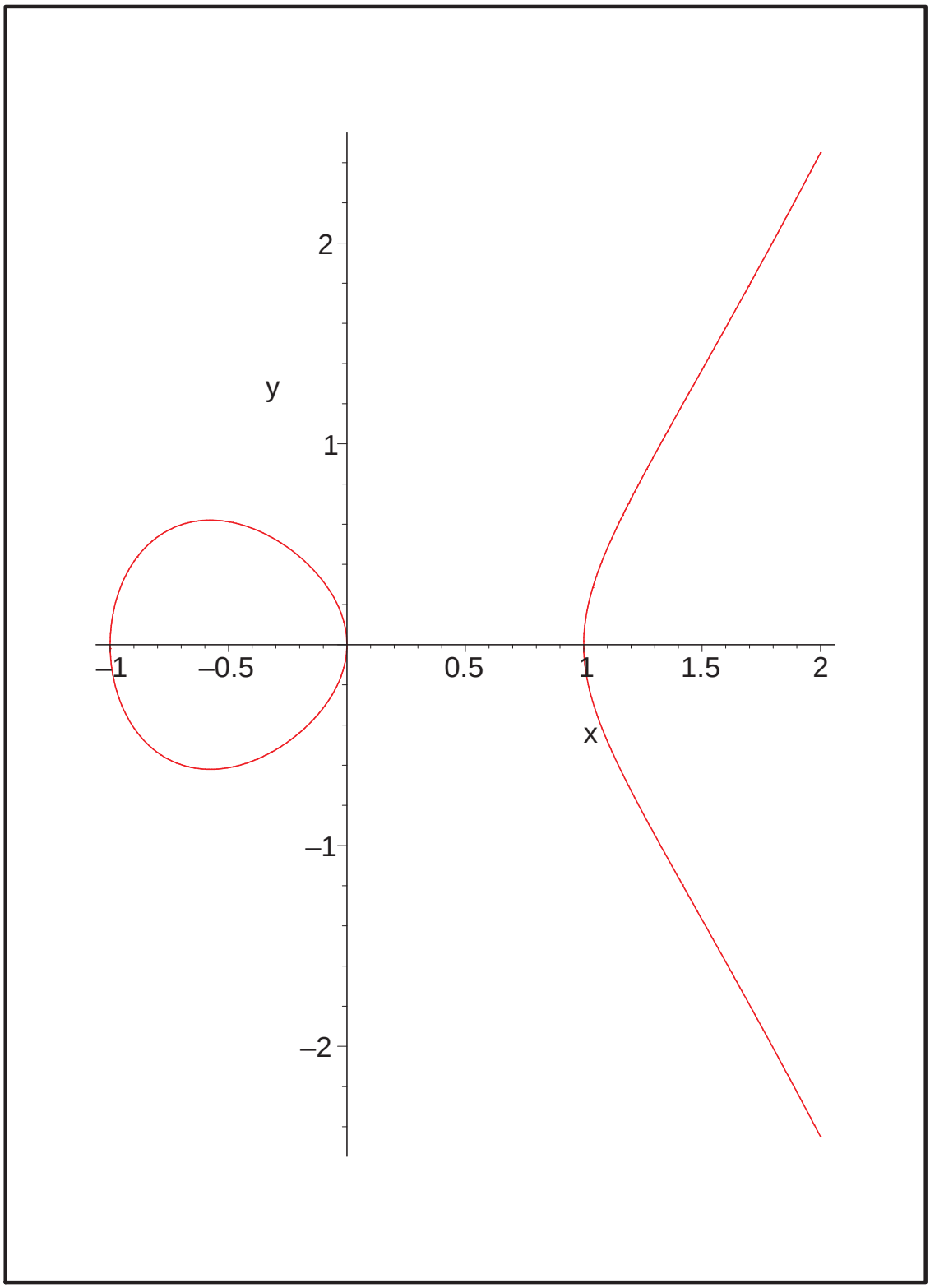

Abbildung 4.1: Darstellung der elliptischen Kurve $y^2 = x^3 - x$ im $\mathbb{R}^2$

Es ist nicht besonders schwierig, die Assoziativität zu zeigen, aber technisch ein wenig aufwendig (die elliptischen Kurven bilden ein gutes Beispiel für Gruppen, bei denen die Assoziativität nicht trivial ist!).

Neutrales Element: Der unendlich ferne Punkt. Also: $P + \infty = \infty + P = P$ für alle $P \in E(\mathbb{F})$.

Inverses Element: Der an der x-Achse gespiegelte Punkt. Falls P = Q, dann müssen wir im ersten Schritt die Tangente von P an E wählen.

Die Punkte der Ordnung zwei auf E sind die Punkte, deren x-Koordinate gleich 0 sind. Die Punkte der Ordnung drei auf E sind die Wendepunkte der Kurve (hier schneidet die Tangente mit Multiplizität drei).

4.2.3 Das Gruppengesetz für $char(\mathbb{F}) > 3$ und $char(\mathbb{F}) = 0$

Seien $P, Q \in E(L)$ Punkte auf der Kurve. Dann lassen sich beide Punkte folgendermaßen addieren (in Abbildung 4.2 veranschaulicht):

Fall 1 - Addition unterschiedlicher Punkte: Verbinde P, Q durch eine Gerade g (blau). Die Steigung von g ist $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$. Der y-Achsenabschnitt $\gamma = y_P - \lambda x_P$. Die Gerade lässt sich dann als

$$g: \; y = \lambda x + \gamma$$

angeben. Der dritte Schnittpunkt der Geraden mit der Kurve ist der Punkt $-(P + Q)$. Von dort kann der Punkt $(P + Q)$ ermittelt werden, in dem man $-(P + Q)$ an der x-Achse spiegelt (braune Linie). Die Koordinaten $(x_3, -y_3)$ des Punktes $-(P + Q)$ erhält man über den Schnittpunkt von g und E:

$$\begin{aligned} y &= \lambda x + \gamma \\ &= \lambda(x - x_1) + y_1 \\ y^2 &= \lambda^2 (x - x_1)^2 + 2\lambda(x - x_1)y_1 + y_1^2 \end{aligned}$$

Dies lässt sich nun in Gleichung 4.6 einsetzen:

$$x^3 - \underline{(\lambda^2(x - x_1)^2)} - 2\lambda(x - x_1)y_1 - y_1^2 + ax + b = 0$$

Diese Gleichung ist nur von x abhängig und lässt sich also als

$$(x - x_1)(x - x_2)(x - x_3) = x^3 - \underline{(x_1 + x_2 + x_3)x^2} + \ldots$$

schreiben. Durch Koeffizientenvergleich erhält man nun:

$$\lambda^2 = x_1 + x_2 + x_3$$

Daraus lässt sich das gesuchte x_3 eindeutig bestimmen. $-y_3$ erhält man durch Einsetzen in die Gleichung von g. Für den gesuchten Punkt $(P + Q)$ gilt also:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

Fall 2 - Addition des selben Punktes (Verdopplung): Bilde die Tangente g (violett) der Kurve in T. Der zweite Schnittpunkt (bzw. dritte, wenn man „T“ doppelt zählt) der Tangente mit der Kurve bildet den Punkt $-2T$. Durch Spiegeln an der x-Achse (die magenta Linie) erhält man den Punkt $2T$. Im Detail berechnet sich also die Verdopplung wie folgt:

Aus der Kurvengleichung

$$y^2 = x^3 + ax + b$$

folgt:

$$y = (x^3 + ax + b)^{\frac{1}{2}}$$

und damit gilt für die Steigung (also die Tangente):

$$\lambda = y' = \frac{1}{2}(x^3 + ax + b)^{-\frac{1}{2}}(3x^2 + a) = \frac{3x^2 + a}{2y}$$

x_3 und y_3 ergeben sich dann aus der eben bestimmten Steigung und den Formeln für den 1.Fall, also:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 = \frac{3x_1^2 + 2}{2y_1} - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 = \frac{3x_1^2 + 2}{2y_1}(x_1 - x_3) - y_1 \end{aligned}$$

4.15 Beispiel. *$K = GF(13)$ und $E : y^2 = x^3 + 2x + 5$ über K*
1) Bestimme die Punkte auf E:

y	y^2
0	*0*
1	*1*
2	*4*
3	*9*
4	*3*
5	*12*
6	*10*
7=-6	*10*
...	*...*

x	$x^3 + 2x + 5$
0	*5*
1	*8*
2	*4*
3	*12*
4	*12*
5	*10*
6	*12*
7	*11*
8	*0*
9	*11*
10	*11*
11	*6*
12	*2*

$$\Rightarrow E = \{(2,2), (2,-2), (3,5), (3,-5), (4,5), (4,-5), (5,6), (5,-6), (6,5), (6,-5), (8,0), 0\}$$

2) Bestimme $2P$ *für* $P = (4, -5) = (4, 8)$

$$\lambda = \frac{3x_1^2 + 2}{2y_1} = \frac{11}{3} = 11 \cdot 9 \equiv 8$$

$\Rightarrow x_3 = \lambda - x_1 - x_2 = 4$ *und* $y_3 = \lambda(x_1 - x_3) - y_1 = 5$*, also:*

$$2P = (4, 5)$$

3) Bestimme $3P$ *für* $P = (4, -5) = (4, 8)$

$3P = 2P + P = (4, 5) + (4, -5)$

Da die beiden Punkte die gleiche X-Koordinate haben, liegt $3P$ *im unendlichen, also* $3P = 0$*. Damit ist auch* $ord(P) = 3$*.*

Fall 3 - Addition von S und $-S$: In Abbildung 4.2 ist der Spezialfall, dass S auf der x-Achse liegt, gewählt. Allgemein gilt für $S+(-S)$, dass man wieder die Verbindungsgerade der beiden Punkte (grün) zieht. Da die Gerade die Kurve erst wieder in dem **0**-Punkt schneidet (also im unendlichen) gilt: $-(S + (-S)) = S + (-S)$. Dies lässt sich dadurch erklären, dass der **0**-Punkt als Gruppenidentität definiert wurde.

Aus den Regeln für die Gruppenoperation folgen zwei Beobachtungen:

- Wenn $P, Q \in E(\mathbb{F})$ gilt, dann gilt auch: $P + Q \in E(\mathbb{F})$ (Abgeschlossenheit)
- Die Gruppenoperation ist assoziativ.

Warnung: Das Verständnis des Gruppengesetzes ist sehr wichtig, da die real verwendeten Gleichungen deutliche komplexer sind aber auf dem Gruppengesetz basieren.

4.16 Bemerkung. *Der Beweis für die Assoziativität ist technisch etwas aufwendiger und wird hier nicht geführt.*

4.2.4 Das Gruppengesetz für $char(\mathbb{F}) = 2$

Im Fall $char(\mathbb{F}) = 2$ kann nicht mehr die vereinfachte Version der Kurvengleichung verwendet werden. Hier muss man auf

$$E: \ y^2 + xy = x^3 + a_2x^2 + a_6$$

mit $a_6 \neq 0$ zurück greifen. Hier gelten einige wichtige Veränderungen im Gruppengesetz:

- Allgemein gilt für Körper mit $char(\mathbb{F}) = 2$ dass Addition und Subtraktion äquivalent sind.
- Die Inversion eines Punktes kann nicht mehr durch die Spiegelung geschehen. Für den Punkt $P = (x_1, y_1)$ gilt $-P = (x_1, x_1 + y_1)$.

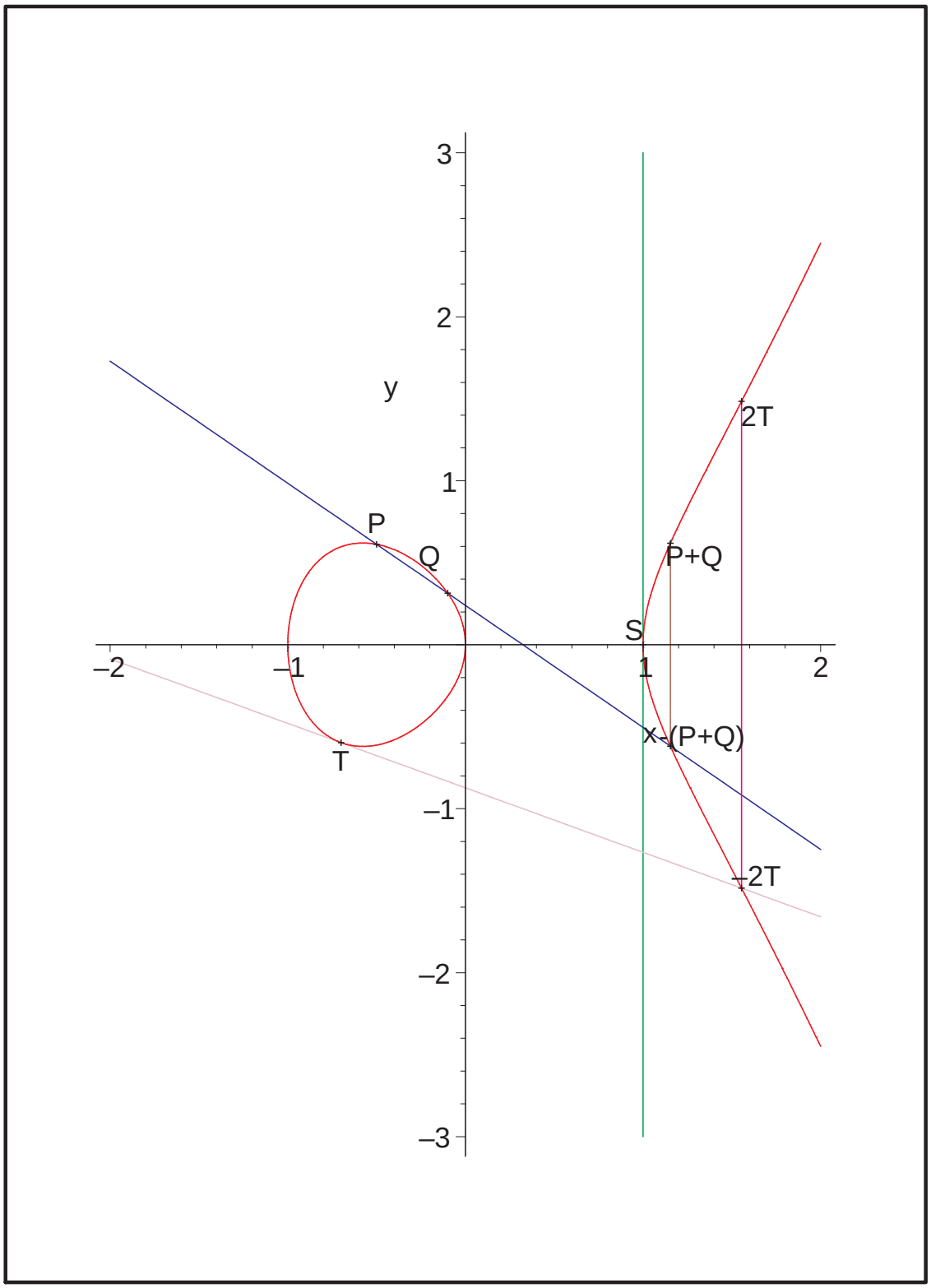

Abbildung 4.2: Demonstration der Gruppenoperation auf einer elliptischen Kurve

- Für die Addition von P, Q mit $P \neq \pm Q$ gilt für die Steigung:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

- Für die Addition $P + P$ gilt für die Steigung:

$$\lambda = x_1 + \frac{y_1}{x_1}$$

- Der Wert für x_3 berechnet sich folgendermaßen:

$$x_3 = \lambda^2 + \lambda + \underbrace{x_1 + x_2}_{=0 \, falls \, P=Q} + a_2$$

- Der Wert für y_3 erhält man entsprechend der obigen Regeln durch die Gleichung:

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

4.17 Bemerkung. *Die Gesetze für die Verdopplung im Fall* $char(\mathbb{F}) = 2$ *sollte man nun selbst entwickeln können.*

4.2.5 Elliptische Kurven über endlichen Körpern

Sei nun $\mathbb{F} = \mathbb{F}_q$ ein endlicher Korper mit q Elementen.

4.18 Beispiel. *Sei* $\mathbb{F} = \mathbb{F}_5$ *der endliche Primkorper mit* 5 *Elementen. Setze* $E : y^2 = x^3 + 1$. *Dann gilt* $E(\mathbb{F}_5) = \{ \infty,\ (0,1),\ (0,4),\ (2,2),\ (2,3),\ (4,0) \}$.

4.19 Bemerkung. *Es gibt nur endliche viele Tupel* (x_0, y_0) *, die der Gleichung der elliptischen Kurve genügen, d.h. die Punktegruppe über einem endlichen Körper ist endlich. Es gilt offenbar* $\#E(\mathbb{F}_q) \leqslant 2q + 1$. *Da nicht jedes Element quadratischer Rest ist, sonst etwa die Hälfte aller Elemente, erwarten wir etwa q Punkte. Tatsächlich a gilt folgender Satz.*

4.20 Satz (Hasse-Weil)**.** *Sei* $\mathbb{F}_q$ *ein Körper mit q Elementen und sei* $E/\mathbb{F}_q$ *eine elliptische Kurve. Dann gilt*

$$|\#E(\mathbb{F}_q) - q - 1| \leqslant 2\sqrt{q} \ .$$

4.3 Effiziente Implementierung der Gruppenoperation

Sei $n \in \mathbb{N}$ und $P \in \mathrm{E(K)}$. Nun stellt sich die Frage, wie man

$$nP = \underbrace{P + P + \ldots + P}_{n \text{ Summanden}}$$

möglichst effizient berechnen kann. Die triviale Methode, n mal P addieren, hat offensichtlich die Komplexität $\Theta(n)$ und ist damit nicht zufriedenstellend. Für den Fall $n = 2^k$ gibt es eine einfache Verbesserung:

$$2^k P : \ P, 2P, 4P, \ldots, 2^{k-1}P, 2^k P \ .$$

Durch Verdoppelung entsteht also nur der Aufwand $\mathbb{O}(\log_2(n))$. Da sich aber jede Zahl n in der Form $n = \sum n_i 2^i$ mit $n_i \in \{0, 1\}$ darstellen lässt, lässt sich folgender Algorithmus (analog zum Square-and-Multiply Algorithmus) angeben:

Algorithm 3. Verdopple-und-Addiere

INPUT: $n = \sum_{i=0}^{\ell-1} n_i 2^i, \ n_i \in \{0, 1\}, \ P \in \mathrm{E(K)}$
OUTPUT: $Q = nP$ (in $\Theta(\log_2(n)) = \Theta(\ell)$ Schritten)

1. $Q \leftarrow 0; //\infty \in \mathrm{E(K)}$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. $\quad Q \leftarrow 2Q$
4. $\quad$ **if** $n_i = 1$ **then** $Q \leftarrow Q + P$
5. **return** (Q)

4.21 Beispiel. *Gesucht:* $37P$
$37 = (100101)_2, \ \ell = 6 \Rightarrow$
$Q \leftarrow 0 \underset{\rightarrow}{,} P, 2P, 4P, 8P \underset{\rightarrow}{,} 9P, 18P, 36P \underset{\rightarrow}{,} 37P$
Die Additionen aus Algorithmus 3 Schritt 4 sind durch die Pfeile gekennzeichnet.

Die Korrektheit des Algorithmuses kann leicht durch Induktion gezeigt werden, wenn man sich klar macht, dass nach jedem Schleifendurchlauf gilt:

$$Q = \left(\sum_{j=i}^{\ell-1} p_j 2^j \right) P$$

4.3.1 Nicht-Angrenzende Form (NAF)

Da die Rechenzeit des Algorithmus deutlich von der Anzahl der Einsen in n (also dem Hamming Gewicht von n) abhängt, kann man nun versuchen, die Anzahl der Einsen zu reduzieren. Die Zahl 31 kann man als $31 = (1111)_2$ oder als $31 = 32 - 1 = (1000 - 1)_2$ darstellen. Aus ästhetischen Gründen schreibt man statt -1 oft $\overline{1}$, also z.B.

$$31 = (1000\overline{1})_2 \ .$$

Daraus folgt:

- $(1111)_2$: $0, P, 2P, 3P, 6P, 7P, 14P, 15P, 30P, 31P \Rightarrow$ #OPs= 8
- $(1000\overline{1})_2$: $0, P, 2P, 4P, 8P, 16P, 32P, 31P \Rightarrow$ #OP's= 6

Die Kodierung mit den Ziffern $\{0, 1, \overline{1}\}$ wird als *Non Adjacent Form* (kurz: NAF; Deutsch: *Nicht-Angrenzende Form*, auch *Nicht-Adjazente Form*) bezeichnet. Die Idee wurde in den 60er Jahren das erste mal von Booth auf Röhrenrechnern in Hardware implementiert. In den 90er Jahren wurde der Algorithmus dann von F. Morain und J. Olivos ([6]) auf elliptische Kurven übertragen.

Um einen effizienten Recodierungsalgorithmus für die NAF zu entwerfen, können wir folgende Beobachtung machen: Eine Zahl $n \in \mathbb{N}$ ist entweder gerade oder es gilt $n \equiv d\, mod(4)$ für ein $d \in D$, wobei $D = \{0, 1, -1\}$ ist.

Dies führt uns zu dem folgenden Recodierungsalgorithmus:

Algorithm 4. Berechnung der NAF

INPUT: $n \in \mathbb{N}$
OUTPUT: $n = \sum n_i 2^i$ mit $n_i \in \{0, 1, -1\}$ und $n_i = 0 \vee n_{i+1} = 0$

1. $z \leftarrow n$
2. $\ell \leftarrow 0$
3. **while** $z \neq 0$ **do**
4. **if** z even **then**
5. $n_\ell \leftarrow 0$
6. **else**
7. Sei $n_\ell \in D$ mit $n_\ell \equiv z\, mod(4)$
8. $z = \frac{z - n_\ell}{2}$
9. **if** $z \neq 0$ **then** $\ell = \ell + 1$
10. **return** $(\sum_{i=0}^{\ell} n_i 2^i)$

Proof. Ohne Beschränkung der Allgemeinheit sei $n \geqslant 0$.

Wir nehmen an, der Algorithmus liefere eine korrekte Entwicklung für $n \leqslant N$ – z.B. dies ist für $N = 1$ einfach verifiziert.

Sei nun $n = N+1$. Wenn n gerade ist, muss $n_0 = 0$ sein (Beweis: reduziere die Gleichheit $\sum_{i=0}^{\ell} n_i 2^i = n \bmod 2$, und beobachte, dass 0 die einzige gerade Ziffer in D ist) und somit liefert der Algorithmus die Ausgabe:

$$(\underbrace{n_\ell \ldots n_1}_{\frac{n-n_0}{2}} \mid \underbrace{n_0}_{=0})$$

wobei die Ziffer bis auf die Minderwertigste eine (per Induktionsannahme valide) Entwicklung von $n/2$ darstellen.

Wenn n hingegen ungerade ist, muss dann auch n_0 ungerade sein, und $n_0 \neq 0$, somit muss sich n, wie folgt zusammen setzten:

$$(\underbrace{n_\ell \dots n_1}_{\frac{n-n_0}{2}} | \underbrace{n_0}_{\in\{1,-1\}}) \ .$$

Da der Algorithmus wählt $n_0 \equiv n \bmod 4$, ist $\frac{n-n_0}{2}$ gerade, und somit ist die minderwertigste Ziffer von $\frac{n-n_0}{2}$ gleich 0, also ist in diesem Fall

$$(\underbrace{n_\ell \dots n_2}_{\frac{n-n_0}{4}} | 0 \underbrace{n_0}_{\in\{1,-1\}}) \ .$$

Somit kann die Korrektheit per Induktion gezeigt werden, da $\frac{n-n_0}{2}$, bzw. $\frac{n-n_0}{4}$ laut der Annahme durch eine korrekte Entwicklung dargestellt wird und $< n$ ist. □

Der Algorithmus 3 kann in leicht abgewandelter Form auch für die NAF verwendet werden:

Algorithm 5. Verdopple-und-Addiere/Subtrahiere

INPUT: $n = \sum_{i=0}^{\ell} n_i 2^i,\ n_i \in \{0, 1, -1\},\ P \in \mathrm{E(K)}$
OUTPUT: $Q = nP$ (in $\Theta(\log_2(n)) = \Theta(\ell)$ Schritten)

1. $Q \leftarrow 0; //\infty \in \mathrm{E(K)}$
2. **for** $i = \ell - 1$ **downto** 0 **do**
3. $\quad Q \leftarrow 2Q$
4. $\quad$ **if** $n_i = 1$ **then** $Q \leftarrow Q + P$
5. $\quad$ **if** $n_i = -1$ **then** $Q \leftarrow Q + (-P)$
6. **return** (Q)

Nun stellt sich die Frage, wie groß der Vorteil durch die NAF ist. Die Zahl z hat genau dann $\ell + 1$ Stellen, wenn $n_\ell = 1$ und $\forall t > l :\ n_t = 0$ gilt. In diesem Fall sagt man, dass die Entwicklung *Länge* $\ell + 1$ hat.

Dafür verwenden wir das Hamminggewicht einer Zahl $n = \sum n_i 2^i = \sum_{i=0}^{\ell} n_i 2^i$. Das Gewicht ist nun definiert als:

$$gew(n) = \#\{n_i | n_i \neq 0\}$$

Die „Dichte“ einer Rekodierung ist gegeben als:

$$d(n) = \frac{gew(n)}{\ell + 1}$$

4.22 Satz. *Der Erwartungswert der Dichte der binären Darstellung einer zufällig gewählten natürlichen Zahl n ist $\mathbb{E}(d(n))_{bin} = \frac{1}{2}$. Der Erwartungswert der Dichte der NAF einer zufällig gewählten natürlichen Zahl n ist $\mathbb{E}(d(n))_{NAF} = \frac{1}{3}$.*

Proof. Folgender Beweis ist eher „intuitiv“ als korrekt. Ein korrekte Beweis ist aber im Prinzip nicht viel anders.

Jede Ziffer der binären Entwicklung einer zufällig gewählten ganzen Zahl ist mit gleicher Wahrscheinlichkeit 0 oder 1 (eine zufällig gewählten ganzen Zahl n ist mit gleicher Wahrscheinlichkeit gerade oder ungerade, also ist der niederwertigste Bit n_0 0 oder 1 mit Wahrscheinlichkeit $\frac{1}{2}$ und $\frac{(n-n_0)}{2}$ ist wieder eine ganze Zahl...).

Jetzt betrachten wir die NAF-Entwicklung einer zufällig gewählten ganzen Zahl n. Mit Wahrscheinlichkeit $1/2$ ist eine Zahl gerade also ist ihre niederwertigste Ziffer $n_0 = 0$, und $\frac{n}{2}$ verhält sich wie eine zufällig gewählte ganze Zahl. Ist die n hingegen ungerade, so ist die niederwertigste Ziffer $n_0 \in \{1, \bar{1}\}$ und $n_1 = 0$nd $\frac{n-n_0}{2}$ verhält sich wie eine zufällig gewählte ganze Zahl.

Man konstruiere eine zufällige NAF, in dem man aus einer Urne mit vier Kärtchen, die mit 0, 0, 01 und $0\bar{1}$ beschriftet sind, t-Mal ein Kärtchen mit zurürlegen zieht. Bei $t/2$ (erwartet) Ziehungen wird eine 0 gezogen und bei den restlichen $t/2$ (erwartet) Ziehungen $0\star$.

Somit ergibt sich eine erwartete Länge von $\frac{t}{2} + 2 \cdot \frac{t}{2} = \frac{3t}{2}$ und ein erwartetes Gewicht von $\frac{t}{2}$. Daher ist die erwartete Dichte der NAF $\mathbb{E}(d(n))_{NAF} = \frac{t/2}{3t/2} = \frac{1}{3}$. □

Durch die NAF können wir den Aufwand also durchschnittlich um $\frac{1}{6}$ senken. Dabei setzen wir voraus, dass es effiziente Verfahren für die NAF-Rekodierung gibt. Dies kann z.B. sehr effizient mit shift-Operationen erledigt werden.

4.3.2 Effizientere Rekodierungen und Skalarmultplikation

Idee: Mit Hilfe einer Darstellung der Form:

$$n = \sum_{i=0}^{\ell} n_i 2^i$$

mit $n_i \in D$, wobei der Ziffernvorrat (oder: die Ziffernmenge) D grösser als $\{0, 1, -1\}$ ist, kann nP noch effizienter berechnet werden.

Um eine solche Darstellung zu erhalten wird eine NAF-ähnliche Recodierung mit dem Ziffernvorrat/Ziffernmenge, die eine der zwei nachfolgenden Kriterien erfüllen muss, angewandt:

1. $D \subseteq \mathbb{N}_0$ also: $D = \{0\} \cup \{1, 3, 5, \ldots, 2^w - 1\}$
2. $D = -D$ also: $D = \{0\} \cup \pm\{1, 3, 5, \ldots, 2^{w-1} - 1\}$

Somit kann der Algorithmus 4 für einen Zeichenvorrat D, der eine der zwei oberen Kriterien erfüllt verallgemeinert werden.

Algorithm 6. Recodierung der w-NAF

INPUT: $n \in \mathbb{N}$
OUTPUT: $n = \sum n_i 2^i$ mit $n_i \in D$ und $n_i = 0 \vee n_i \neq 0 \Rightarrow n_i = n_{i+1} = \ldots = n_{i+w-1} = 0$

1. $z \leftarrow n$
2. $\ell \leftarrow 0$
3. **while** $z \neq 0$ **do**
4. **if** z even **then** $n_\ell \leftarrow 0$
5. **else**
6. Sei $n_\ell \in D$ mit $n_\ell \equiv z \, mod(2^w)$
7. $z = \frac{z - n_\ell}{2}$
8. **if** $z \neq 0$ **then** $\ell = \ell + 1$
9. **return** $(\sum_{i=0}^{\ell} n_i 2^i)$

Der Korrektheitsbeweis kann analog zu dem Korrektheitsbeweis der 2-NAF Recodierung geführt werden.

Um eine effiziente Punktverdopplung (Skalarmultiplikation) zu realsieren, kann ein Algorithmus, der eine spezielle Form des k-ary Algorithmus für elliptische Kurven darstellt, benutzt werden.

Algorithm 7. Skalarmultiplikation

INPUT: $n = \sum n_i 2^i$ mit $n_i \in D$ und $P \in E(\mathbb{F})$
OUTPUT: $Q = nP$

1. Precompute: Fur alle $d \in D \cap \{1, 2, 3, \ldots, \infty\}$ berechne dP
2. $Q \leftarrow 0$
3. **for** $i = l$ **downto** 0 **do**
4. $Q \leftarrow 2Q$
5. **if** $n_i \neq 0$ **then** $q \leftarrow Q + n_i P$
6. **return** (Q)

Die Komplexität des Algorithmus ist 2^{w-2} Gruppenoperationen in Vorberechnung um die Lookup Table zu erstellen und noch weitere $\frac{\ell}{w+1}$, die der erwarteten Dichte $\frac{1}{w+1}$ der w-NAF entspricht (der Beweis davon ist sehr ähnlich zu dem vom Satz 4.22). Hierbei gilt: $\ell \approx \log_2(|G|)$.

Wenn die Effizienz des Algorithmus gesteigert werden soll, gilt es also $2^{w-2} + \frac{\ell}{(w+1)}$ zu minimieren.

Für w gilt $w \approx \log_2(\ell) - 2\log_2(\log_2(\ell))$ (cfr. Knuth, ACP Band 2).

4.4 Public Key Kryptographie: Protokolle

4.4.1 Schlüsselaustausch nach Diffie-Hellman

Der Diffie-Hellman-Schlüsselaustausch ist kein Verschlüsselungsverfahren, sondern beschreibt die Möglichkeit, Schlüssel sicher über unsichere Kanäle, wie Telefonleitungen, anzeigen in Zeitungen, Email, Webseiten, usw. auszuhandeln. Hierbei handelt es sich um Schlüssel, wie sie in der Kryptographie verwendet werden.

Das Verfahren ist ein typisches Public Key (asymmetrisches) Verfahren, und der resultierende Schlüssel ist normalerweise in symmetrischen Verfahren (Secret Key Verfahren) verwendet, denn die symmetrischen Verschlüsselungsverfahren in der Regel viel schneller als die Asymmetrischen sind.

Der Algorithmus wurde von Martin Hellman gemeinsam mit Whitfield Diffie und Ralph Merkle an der Universität von Stanford (Kalifornien) entwickelt und 1976 veröffentlicht.

Wie erst 1997 bekannt wurde, hatte das britische Government Communications Headquarters (GCHQ) schon in den 1960er Jahren den Auftrag erteilt, aufgrund der hohen Kosten bei der damals üblichen Schlüsselverteilung einen anderen Weg für die Schlüsselverteilung zu finden. Die von James Ellis, Clifford Cocks und Malcolm Williamson geäußerten Ideen ähnelten dem Diffie-Hellmann-Verfahren. Das GCHQ hat einerseits wegen der Geheimhaltung, andererseits wegen des für die Briten aus Sicht der frühen 1970er Jahre fraglichen Nutzens nie ein Patent beantragt.

Die Sicherheit des Verfahrens hängt von der Schwierigkeit, diskrete Logarithmen zu berechnen, ab, obwohl die Schwierigkeit, das Protokoll zu knacken, ist nicht offensichtlich zum DLP äquivalent.

Beschreibung

Hier wird nur eine „bare bones“-Variante des Protokolls beschrieben, die in der Tat unsicher ist, weil ein Angreifen kann sich eindringen und die Rolle von einem der zwei Kommunikationspartners übernehmen. Mathematisch ist aber das System zu den verbesserten Versionen äquivalent.

Zwei Kommunikationspartner wollen ein symmetrisches Kryptosystem über ein unsicheres Medium einsetzen, dafür fehlt ihnen einen gemeinsamen geheimen Schlüssel, der über das unsichere Medium nicht unverschlüsselt übertragen werden soll. Der Diffie-Hellman-Schlüsselaustausch bietet eine Lösung zu diesem Problem.

1. Die Kommunikationspartner (Alice und Bob) einigen sich zunächst auf eine zyklische Gruppe G (additiv geschrieben) der Ordnung ℓ und ein erzeugendes Element γ der

Gruppe G. Diese Parameter (G, ℓ, γ) müssen nicht geheim bleiben, können also insbesondere auch über ein unsicheres Medium übertragen werden.

2. Beide Kommunikationspartner erzeugen jeweils eine geheim zu haltende Zufallszahl a bzw. b (für Alice und Bob) aus der Menge $\{1, ..., p-2\}$. a und b werden nicht übertragen, bleiben also dem jeweiligen Kommunikationspartner, aber auch potenziellen Lauschern unbekannt.

3. Alice und Bob berechnen $A = a \cdot \gamma$ bzw. $B = b \cdot \gamma$. Nun werden A und B über das unsichere Medium übertragen.

4. Alice berechnet nun $K = a \cdot B = (ab) \cdot \gamma$, und Bob berechnet $K' = b \cdot A = (ab) \cdot \gamma$. Das Ergebnis $K = K'$ ist für beide Partner gleich und kann (möglicherweise nach einer einfachen Bearbeitung) als Schlüssel für die weitere Kommunikation verwendet werden.

Sicherheitsbetrachung

Um das System zu knacken, ein Angreifer muss in der Lage sein, aus γ, $a \cdot \gamma$ und $b \cdot \gamma$ das Gruppenelement $(ab) \cdot \gamma$ zu rekonstruieren. Das Problem heisst Diffie-Hellman Problem.

Falls es gelint, aus γ und $a \cdot \gamma$ die Zahl a zu finden (Discrete Logarithm Problem), dann, kann man auch das Diffie-Hellman Problem lösen.

4.4.2 ECDSA

Wir beschreiben nun eine Variante des Signaturalgorithmus DSA für elliptische Kurven (ECDSA). Hierbei steht ECDSA für Elliptic Curve Digital Signature Algorithm. Dieser Algorithmus ist als IEEE-Standard eingeführt worden.

ECDSA:
Parameter (öffentlich): Kurve $E(\mathbb{F}_p)$, Punkt $P \in E(\mathbb{F}_p)$ der Ordnung ℓ (ℓ ist eine grosse Primzahl).
Privater Schlüssel: eine ganze Zahl x im Intervall $[2 \ldots \ell - 2]$
Öffentlicher Schlüssel: der Punkt Q mit $xP = Q$.
$H(\cdot)$ ist eine vorhandene, feste Hashfunktion.

Beschreibung

Signaturerzeugung:
Eingabe: eine Nachricht m, Alices privater und öffentlicher Schlüssel x und Q.
Ausgabe: Alices elektronische Signatur (r, s)

1. A(lice) wählt $k \in [2 \ldots \ell - 2]$ zufällig.

2. A berechnet $k \cdot P = (x_1, y_1)$ und $r = x_1 \bmod \ell$. Falls $r = 0$ dann geht sie zu Schritt 1 zurück, da sonst die Signatur s, die in Schritt 3 berechnet wird, nicht vom privaten Schlüssel x abhängig wäre.

3. A berechnet $k^{-1} \bmod \ell$ und den Hashwert $H(m)$ der Nachricht m. Dann bildet sie $s = k^{-1}(H(m) + xr) \bmod \ell$.

4. Falls $s = 0$, beginnt sie wieder bei Schritt 1. Sonst sendet A die Nachricht m zusammen mit der Signatur (r, s) an B(ob).

Signaturverifikation:
EINGABE : Die Nachricht m, eine Signatur (r, s), Alices öffentlicher Schlüssel Q
AUSGABE : Die Signatur ist „korrekt“ oder „falsch“

1. B prüft, ob $r, x \in [1 \ldots \ell - 1]$.

2. B berechnet $w = s^{-1} \bmod \ell$ und $H(m)$. Dann bildet er $u_1 = H(m)w \bmod \ell$ und $u_2 = rw \bmod \ell$.

3. B ermittelt $u_1 \cdot P + u_2 \cdot Q = (x_0, y_0)$.

4. Bob entscheidet sich für „korrekt“(also akzeptiert die Signatur als gültig) genau dann, wenn $x \bmod \ell = r$ ist, sonst für „falsch“.

Zur Korrektheit und Sicherheit

Zur Korrektheit: Wir nehmen an, dass Alice die Signaturerzeugung korrekt ausgeführt hat. Dann berechnet Bob im dritten Schritt der Signaturverifikation

$$u_1 P + u_2 Q = (H(m)w \bmod \ell)P + (rw \bmod \ell)Q = (x_0, y_0) \ .$$

Nun gilt aber $x \cdot P = Q$. Also erhält Bob

$$\begin{aligned}(H(m)w \bmod \ell)P + (rw \bmod \ell)xP \\ &= ((H(m)w + rwx) \bmod \ell)P \\ &= (w(H(m) + rx) \bmod \ell)P = kP \ .\end{aligned}$$

Deshalb mußx$_0 = r \bmod \ell$ gelten. Falls ein beliebiger Benutzer C den diskreten Logarithmus berechnen kann, erhält er aus P und Q den geheimen Schlussel x und kann damit Signaturen von A fälschen.

4.5 Sicherheit

4.5.1 Das diskrete Logarithmusproblem

Unter dem Problem des diskreten Logarithmus (kurz: DLP, aus dem englischen *discrete logarithm problem*) auf der elliptischen Kurve versteht man die folgende Fragestellung: Gegeben eine über einem endlichen Körper $\mathbb{F}_q$ definierte Kurve und zwei Punkte P, Q aus der Punktegruppe $E(\mathbb{F}_{q^k})$ über der Körpererweiterung $\mathbb{F}_{q^k}$, finde eine ganze Zahl $r \in \mathbb{Z}$ derart, dass $r \cdot P = Q$, falls ein solches r existiert.

Die Annahme der Existenz von r ist in den praktischen Anwendungen keine Beschränkung, denn P ist normalerweise das erzeugende Element einer (selbstverständlich zyklischen) Untergruppe von $E(\mathbb{F}_{q^k})$ und $Q \in \langle P \rangle$.

Im allgemeinen Fall muss die Zahl r nicht unbedingt existieren, z.B. wenn $Q \notin \langle P \rangle$. Die Existenz von r lässt sich aber (mit der so-genannten Weil-Paarung) leicht überprüfen.

Im folgenden werden wir immer annehmen, dass $Q \in \langle P \rangle$ ist.

Die Tripel (E, P, Q) bildet eine Instanz des diskreten Logarithmusproblems.

Eine Instanz (E, P, Q) des diskreten Logarithmusproblems lässt sich mit dem square-and-multiply-Algorithmus leicht berechnen.

Für vorsichtig gewahlte Kurven (siehe § **??** unten für die wichtigsten Ausnahmen) hat aber der beste bekannte Algorithmus zum lösen des DLPs Komplexität $O(\sqrt{\ell})$, wobei ℓ der größte Primfaktor der Gruppenordnung ist (Algorithmus von Silver-Pohling-Hellman - im Grunde genommen, eine Anwendung des chinesischen Restsatzes).

4.5.2 Parameterwahl, Diffie-Hellman Problem

Es ist offensichtlich, dass man für G eine Untergruppe G der Ordnung ℓ der Gruppe der $\mathbb{F}_q$-rationalen Punkten einer elliptischen Kurve, wobei ℓ eine Primzahl ist, wählen kann, und für γ ein erzeugendes Element von G. Falls $G = E(\mathbb{F}_p)$, sind alle nichttrivialen Elementen erzeugende Elemente der Gruppe. Falls $c = [E(\mathbb{F}_p) : G]$ eine kleine Zahl ist (kleiner als ℓ), ist offensichtlich $c \cdot \eta$, $\eta \in E(\mathbb{F}_p)$, fast immer ein erzeugendes Element von G.

Falls das symmetrische Verfahren Schlüssel von n bits braucht, kann man elliptische Kurven mit $q \approx 2^{n+\delta}$ (wobei δ eine kleine ganze Zahl ist) benutzen, und der Schlüssel für das symmetrische Verfahren besteht aus n bits aus der x-Koordinate vom Element (Punkt!) K. Man kann $\delta = 0$ wählen, oder, z.B. $\delta = 2$ und die mehrwertige und mindestwertige Bits der x-Koordinate ignorieren.

Falls das DLP auf E leicht zu berechnen ist, ist das System unsicher. Wer γ, $A = a \cdot \gamma$, $B = b \cdot \gamma$ kennt, kann a berechnen, und somit auch $K = a \cdot B$.

Das DH-System beruht eigentlich auf die Sicherheit vom folgenden Problem: Gegeben γ, $A = a \cdot \gamma$, und $B = b \cdot \gamma$, berechne $K = (ab) \cdot \gamma$. Das Berechnen von a und b ist hier nicht Teil des Problems. Dies Problem heisst das Diffie-Hellman-Problem (DHP). Im Allgemeinen sind das DLP und das DHP nicht äquivalent. Für elliptische Kurven die zwei Probleme sine aber äquivalent:

- U.M. Maurer and S. Wolf: Diffie-Hellmann oracles: in Advances in cryptology - CRYPTO '96, Lect. Notes in Computer Science Vol. 1109(1996) pp. 268-282, Springer-Verlag

- A. Muzereau, N.P. Smart, F. Vercatueren: The equivalence between the DHP and DLP for elliptic curves used in practical applications, LMS J. Comput. Math., 7(2004), 50-72.

4.5.3 Schwache elliptische Kurven

Die bekannten Familien:

1. Frey-Rück-Reduktion (durch die Tate-Paarung) und MOV-Reduktion (hier MOV = Menezes-Okamoto-Vanstone, durch die Weil-Paarung): Sei $E(\mathbb{F}_q)[n]$ die Untergruppe von $E(\mathbb{F}_q)$, die aus der über $\mathbb{F}_q$ rationalen Punkte auf E besteht, deren Ordnung die Zahl n teilt.

Die FR- oder MOV-Reduktion ist erfolgeich, falls $n \mid q-1$. Falls die Gruppe der n-ten Einheitswurzeln μ_n in $\mathbb{F}_q$ liegt, also $n \mid q-1$, dann gibt es eine Abbildung Φ_n von Tupeln (P,Q) mit $P \in E(\mathbb{F}_q)[n]$ und $Q \in E(\mathbb{F}_q)$ in die Gruppe der n-ten Einheitswurzeln μ_n . Diese hat folgende Eigenschaften:

(a) Sie ist $\mathbb{Z}$-linear in beiden Komponenten, d.h. $\Phi_n(P+P',Q) = \Phi_n(P,Q)\Phi_n(P,Q)$ und $\Phi_n(P,Q+Q') = \Phi_n(P,Q)\Phi_n(P,Q')$.

(b) Zu einer zyklische Untergruppe von $E(\mathbb{F}_q)$ der Ordnung n existiert ein Punkt $P' \in E(\mathbb{F}_q)$ mit $\left\{\Phi_n(P,P'); P \in E(\mathbb{F}_q)[n]\right\} = \mu_n$.

(c) Die Paarung Φ_n läßt sich in $O(\log q)$ elliptischen Operationen (d.h. Punktadditionen oder -verdopplungen) berechnen. Der Punkt P' aus (b) läßt sich in probabilistisch polynomialer Zeit in $\log q$ finden. Falls die Ordnung der Kurve prim ist, können wir ihn sogar direkt angeben.

2. Sei nun $(E(\mathbb{F}_q), P, Q)$ eine Instanz eines DLPs mit $\operatorname{ord} P = n$ und $Q = r \cdot P$ für ein unbekanntes $r \in \mathbb{Z}/n\mathbb{Z}$. Nun sei $P' \in E(\mathbb{F}_q)$, so dass $\alpha = \Phi_n(P,P')$ eine primitive n-te Einheitswurzel ist. (Diesen Punkt gibt es nach (b)). Wir setzen $\beta = \Phi_n(Q,P')$, und wir haben aufgrund der $\mathbb{Z}$-Linearitat: $\beta = \Phi_n(Q,P) = \Phi_n(rP,P') = \Phi_n(P,P')^r = \alpha^r$. Somit haben wir das DLP auf der elliptische Kurve auf ein Problem in dem darunterliegenden Körper reduziert. Da es für das dDLP in $\mathbb{F}_q$ subexponentieller Algorithmen gibt, können wir dann den diskreten Logarithmus auf der elliptischen Kurve in subexponentieller Zeit berechnen.

Falls $n \nmid q-1$, aber $n \mid q^k-1$ für kleines k , dann können wir die Punktegruppe $E(\mathbb{F}_{q^k})$ der elliptischen Kurve über dem Erweiterungskörper $\mathbb{F}_{q^k}$ betrachten. Wir definieren dann die Paarung auf Tupeln (P,Q) mit $P \in E(\mathbb{F}_{q^k})[n]$ und $Q \in E(\mathbb{F}_{q^k})$, und reduzieren das DLP auf $E(\mathbb{F}_q)$ mit der Abbildung Φ_n auf das DLP in $\mathbb{F}_{q^k}$.

Damit das diskrete Logarithmusproblem nicht effizient in einen endlichen Körper reduziert werden kann, fordern wir also, dass der große Primfaktor p der Gruppenordnung keine der Zahlen $q^k - 1$ mit $k \leqslant (\log q)^2$ teilt. Damit sind die sogenannten supersingulären Kurven für kryptographische Anwendung uninteressant, da hier $k \leqslant 6$ gilt.

Für eine zufällig gewähltes Paar (p, E) mit p eine Primzahl und E eine elliptische Kurve über $\mathbb{F}_p$ mit $\#E(\mathbb{F}_p) = \ell$ schätzen Koblitz und Balasubramanian die Wahrscheinlichkeit für $\ell \mid p^k - 1$ für ein $k \leqslant (\log p)^2$ ab. Für eine zufällig gewählte gewöhnliche Kurve ist die Wahrscheinlichkeit für ein kleines k verschwindend gering.

3. Reine anomale Kurven: Falls $\#E(\mathbb{F}_p) = p$ gilt, gibt es sogar Algorithmen, die den diskreten Logarithmus in der Punktegruppe $E(\mathbb{F}_p)$ in linearer Zeit in $\log p$ berechnen. Bei Kurven, für die $\#E(\mathbb{F}_p) = p$ gilt, können wir das Logarithmusproblem auf einer Kurve über $\mathbb{Q}_p$ (der Ring der rationalen Zahlen deren Nennern nicht durch p teilbar sind) „liften“. Diese Liftung läßt sich sehr effizient berechnen da es genügt alle auftretenden Zahlen, insbesondere die Koordinaten der Punkte P und Q, bis zu einer Genauigkeit modp^2 zu berechnen. Das diskrete Logarithmusproblem lässt sich schließlich auf ein DL-Problem in der additiven Gruppe $\mathbb{F}_p$ reduzieren, das trivial mit dem euklidischen Algorithmus lösbar ist. Diese Kurven bieten also überhaupt keine Sicherheit. Bei der Wahl einer über einem großen Primkörper $\mathbb{F}_p$ definierten Kurve müssen wir deshalb darauf achten, dass $\#E(\mathbb{F}_p) \neq p$ ist.

4. Weil-Restriktion: Einige spezielle Kurven über Körpern $\mathbb{F}_{p^n}$ mit einer zusammengesetzten Zahl n sind ebenfalls unsicher (also wähle $n = 1$ oder p klein und n a großund prim).

Die Weil-Restriktion arbeitet wie folgt. Sei E eine elliptische Kurve über $\mathbb{F}_{q^k}$. Die Körpererweiterung $\mathbb{F}_{q^k}$ ist ein Vektorraum der Dimension k über $\mathbb{F}_q$, also kann man eine Basis für $\mathbb{F}_{q^k} / \mathbb{F}_q$ wählen und die Elemente von $\mathbb{F}_{q^k}$ mit k-tuplen von Elementen aus $\mathbb{F}_q$ identifizieren. Dies gilt selbstverstüandlich für die Koordinaten der Punkte auf $E(\mathbb{F}_{q^k})$, also kann man den Punkten auf $E(\mathbb{F}_{q^k})$ Vektoren der Länge $2k$ über $\mathbb{F}_q$ zuordnen. Die Menge W aller solchen Vektoren erbt eine Gruppenstruktur von $E(\mathbb{F}_{q^k})$, und W läßt sich auch als *algebraische Varietät* in $\mathbb{F}_{q^{2k}}$ schreiben: die „Koordinaten“ über $\mathbb{F}_q$ erfüllen Gleichungen, die man in einigen Fällen effizient vereinfachern kann.

Es existieren „Index-Berechnung“ (Index calculus) Algorithmen die das DLP auf solchen Varietäten in Zeit $O(q^2)$ lösen können, und da für die Ordnung ℓ der für kryptographische Zwecke verwendeten Gruppe $\ell \approx q^n$ gilt, sind solche Algorithmen viel schneller als $O(\sqrt{\ell})$.

Neue Varianten von diesem Angriff werden ständig erfunden, die neue Mengen von Beispielkurven knacken können, also sollte man Kurven über Körper $\mathbb{F}_{p^k}$ mit k zusammengesetzt oder mit $k > 1$ und p „mittelgross“ vermeiden.

4.5.4 Wahl der Kurve für kryptographische Anwendungen

Wir fassen die Anforderungen an die Gruppenordnung der elliptischen Kurve zusammen:

1. Die Gruppenordnung soll durch eine große Primzahl ℓ geteilt werden, die die Laufzeit aller Algorithmen für das DLP in beliebigen Gruppen unzulässig lang macht: diese Algorithmen haben alle eine Laufzeit $O(\sqrt{\ell})$, und heute gilt als „unmöglich“ eine Berecchnung von $\approx 2^{80}$ Operationen. Die Primzahl ℓ sollte dann mindestens 160 Bits haben, mir anderen Worten $\ell \approx 2^{160}$.

2. Die Kurve sollte entweder über einem großen Primkorper $\mathbb{F}_q = \mathbb{F}_p$ definiert sein oder über einem Körper $\mathbb{F}_q = \mathbb{F}_{p^n}$ mit sehr kleiner Charakteristik (z.B. $p = 2$) und n eine große Primzahl.

3. Für die Primzahl ℓ – die Ordnung der für kryptographischen Zwecken verwendeten Gruppe – sollte $n \nmid q^k - 1$ für alle $k \leqslant (\log q)^2$ gelten. Insbesondere sollte die Kurve nicht supersingulär sein (für diese Kurven ist $k \leqslant 6$).

4. Falls wir eine über einem großen Primkörper $\mathbb{F}_p$ definierte Kurve wählen, fordern wir, dass $p \neq \#E(\mathbb{F}_p)$ gilt.

Um zu überprüfen, ob eine elliptische Kurve den gestellten Anforderungen genügt, müssen wir ihre Gruppenordnung kennen. Es ist prinzipiell möglich, über einem endlichen Körper $\mathbb{F}_q$ definierte Kurven zufällig zu wählen und deren Punkte zu zählen, da dafür Algorithmen existierten, die polynomiell in $\log q$ laufen. Der Algorithmus zum Punktezählen von Schoof, samt Verbesserungen von Elkies und Atkin, hat eine Laufzeit von $O(\log q)^6$, und ist geeignet zum Punktezählen auf Kurven mit ungefähr 240 Bits. Noch schneller kann man auf Kurven kleiner Charakteristik die Punkte zählen. Eine Alternative besteht darin, eine elliptische Kurve mit gewvunschter Gruppenordnung zu konstruieren oder spezielle Kurven zu nehmen, bei denen die Punkteanzahl auf der Hand liegt und kein Zählalgorithmus nötig ist: diese Methode heist die CM-Methode, wobei CM = complex multiplication (Komplexe Multiplikation).

Kapitel 5

Generischen Gruppen

In diesem Kapitel wollen wir folgenden Satz beweisen.

5.1 Theorem. *(Informell) Ein Algorithmus, der nur die Gruppenoperation benutzt, kann diskrete Logarithmen nicht effizient berechnen.*

Am besten wäre natürlich ein Satz mit einer Aussage wie "Diskrete Logarithmen sind schwer zu berechnen", da aber niemand weiß, wie sich ein solcher Satz beweisen ließe, muss man die Klasse der Algorithmen, die man betrachtet einschränken. Eine mögliche, und durchaus sinnvolle, Einschränkung ist zu verlangen, dass die Algorithmen für jede Gruppe funktionieren sollen und genau um diese Einschränkung soll es im Folgenden gehen.

Im weiteren Verlauf behandeln wir nur zyklische Gruppen primer Ordnung. Sei G eine endliche zyklische Gruppe und $|G| = p$ prim. Es gilt $G \cong (\mathbb{Z}_p, +)$ und für unseren Zweck können wir uns im folgenden für konkrete Operationen auf die Gruppen $(\mathbb{Z}_p, +)$ beschränken.

5.1 Ausschließliche Verwendung von Gruppenoperationen

Man muss sicher stellen, dass der Algorithmus A zum Berechnen von diskreten Logarithmen nur Gruppenoperationen nutzen kann. Zu diesem Zweck definiert man ein Orakel, welches die Gruppenoperationen für A berechnet. Dazu erzeugt das Orakel zunächst eine interne Liste mit Gruppenelementen aus $\mathbb{Z}_p$.

Initialisierung des Orakels

Das Orakel bekommt als Input einen Wert $x \in_R \mathbb{Z}_p$ und wählt zufällige Bitfolgen $\{0,1\}^{\lceil \log_2 p \rceil}$.

Notation: Für ein $x \in \mathbb{Z}_p$ sei $\sigma(x) \in \{0,1\}^{\lceil \log_2 p \rceil}$ das vom Orakel gewählte *Encoding* von x.

Das Orakel gibt nach der Initialisierung aus:

$$\sigma(0), \sigma(1), \sigma(x)$$

Interaktion mit dem Orakel

Der Algorithmus A kann Anfragen an das Orakel stellen. Bei Eingabe von $\sigma(x)$ und $\sigma(y)$ testet das Orakel zunächst, ob $x + y$ bereits in der internen Liste vorhanden ist. Wenn ja, gibt das Orakel das entsprechende Encoding $\sigma(x + y)$ aus seiner Liste aus. Wenn $x + y$ noch nicht in der Liste enthalten war, wählt das Orakel ein neues Encoding für $x + y$, fügt $x + y$ in die interne Liste ein und gibt schließlich $\sigma(x + y)$ aus.

Die Berechnung von Inversen wird analog gehandhabt.

Das Orakel stellt sicher, dass gleiche Elemente gleiche Encodings und unterschiedliche Elemente unterschiedliche Encodings erhalten.

Berechnen von diskreten Logarithmen mit Orakel

Ein Algorithmus A mit Zugriff auf das Orakel soll den diskreten Logarithmus von $x \in (\mathbb{Z}_p, +)$ zur Basis 1 berechnen. Das bedeutet aber nichts anderes als das A den Wert x berechnen soll.

Eine Lösung für dieses Problem wäre z.B. der Baby-Step-Giant-Step Algorithmus (s. Kapitel 3.1.2).

Im Folgenden wird gezeigt, dass das Problem nicht besser als mit Hilfe des Baby-Step-Giant-Step Algorithmus zu lösen ist.

5.2 Simulation des Orakels

Die Idee besteht darin, das Orakel durch ein anderes zu ersetzen, dessen Berechnungen unabhängig von der Eingabe x sind. Dann ist intuitiv klar, dass ein Algorithmus, der dieses Orakel zur Berechnung von diskreten Logarithmen verwendet, x nicht ermitteln kann.

Man muss jedoch sicher stellen, dass der Algorithmus nicht merkt, dass er nicht mit dem ursprünglichen Orakel interagiert.

Das Orakel, welches seine Berechnungen unabhängig von x durchführt, nennt man Simulations-Orakel, da es versucht, das ursprüngliche Orakel perfekt zu simulieren, so dass ein Algorithmus A im Idealfall nicht unterscheiden kann, ob er mit dem simulierten oder dem echten Orakel interagiert.

Das Simulations-Orakel rechnet mit Polynomen in $\mathbb{Z}_p[X]$. Es erzeugt eine interne Liste mit Polynomen, welche bei Anfragen addiert oder invertiert (additiv) werden. Das Simulations-Orakel wählt für diese Polynome analog zum echten Orakel Encodings, wobei gleiche Polynome wiederum gleiche Encodings und unterschiedliche Polynome unterschiedliche Encodings erhalten.

Ein Algorithmus A ist erfolgreich, wenn er

(E1) x berechnen oder

(E2) unterscheiden kann, ob er mit dem echten oder dem Simulations-Orakel interagiert, d.h., ein Simulationsfehler aufgetreten ist.

Die Wahrscheinlichkeit, dass ein Algorithmus mit Hilfe des echten Orakels erfolgreich ist, ist *kleiner* als die Wahrscheinlichkeit, dass er erfolgreich ist, wenn er mit dem Simulations-Orakel interagiert. Dies sollte intuitiv klar sein, da sich beide Orakel, solange kein Simulationsfehler auftritt, nach außen gleich verhalten, und damit auch die Erfolgswahrscheinlichkeit bei beiden gleich ist; wenn ein Simulationsfehler auftritt hat der Algorithmus im simulierten Modell gewonnen.

5.3 Erfolgswahrscheinlichkeit

Im Folgenden wird die Erfolgswahrscheinlichkeit eines Algorithmus betrachtet werden.

Betrachtungen zu E1

Da die Berechnungen des Simulations-Orakels unabhängig von x sind, kann der Algorithmus A den diskreten Logarithmus, d.h. den Wert x nur raten. Die Wahrscheinlichkeit, dass er richtig rät, ist

$$\text{Prob}(E1) = \frac{1}{|G|} = \frac{1}{p}$$

Betrachtungen zu E2

Der Algorithmus kann genau dann einen Unterschied zwischen dem Original- und dem Simulations-Orakel feststellen, wenn für zwei Polynome P_1, P_2 in der internen Liste mit $P_1 \neq P_2$ gilt

$$P_1(x) = P_2(x)$$

Wenn dieser Fall eintritt, hätte das Original-Orakel zwei gleiche Encodings gewählt.

Unter der Annahme, dass der Algorithmus A höchstens m Anfragen an das Orakel gestellt hat, hat die interne Liste des Orakels eine Länge von höchstens m Einträgen. Es gilt

$$P_1(x) = P_2(x) \qquad \Leftrightarrow \qquad (P_1 - P_2)(x) = 0$$

Da alle Polynome der Liste linear, d.h., vom Grad $\leqslant 1$ sind, folgt

$$\text{grad}(P_1 - P_2) \leqslant 1$$

Damit hat $P_1 - P_2$ höchstens eine Nullstelle. Daraus folgt

$$\text{Prob}\left((P_1 - P_2)(x) = 0\right) \leqslant \frac{1}{|G|} = \frac{1}{p}$$

Die Gesamtwahrscheinlichkeit für E2 ergibt sich somit zu

$$\begin{aligned} \text{Prob}(E2) &\leqslant \sum_{P_i,P_j \in L, P_i \neq P_j} \text{Prob}\left((P_1 - P_2)(x) = 0\right) \\ &\leqslant m(m-1)\frac{1}{p} \end{aligned}$$

Wobei L die interne Liste des Orakels bezeichnet wobei $|L| \leqslant m$ gilt.

Die Wahrscheinlichkeit, dass der Algorithmus unter Verwendung des Simulations-Orakels gewinnt, ist somit

$$\begin{aligned} &\leqslant \text{Prob}(E1) + \text{Prob}(E2) \\ &= \frac{1}{p} + \frac{1}{p}m(m-1) \end{aligned}$$

Folglich ist die Wahrscheinlichkeit $\text{Prob}(Succ)$ mit höchstens m Gruppenoperationen einen diskreten Logarithmus zu berechnen

$$\text{Prob}(Succ) \leqslant \frac{1}{p} + \frac{1}{p}m(m-1)$$

Um einen diskreten Logarithmus mit einer Wahrscheinlichkeit $\text{Prob}(Succ) \geqslant \frac{1}{2}$ zu berechnen, muss gelten

$$\frac{1}{2} \leqslant \frac{1}{p} + \frac{1}{p}m(m-1) \approx \frac{m^2}{p}$$

Damit erhalten wir folgendes Theorem:

5.2 Theorem. *Die Anzahl m der benötigten Gruppenoperationen, um den diskreten Logarithmus in generischen Gruppen zu lösen, beträgt:*

$$m \geqslant \frac{1}{\sqrt{2}}\sqrt{p}$$

Damit wurde gezeigt, dass (bis auf einen kleinen Faktor) der Baby-Step-Giant-Step Algorithmus der beste Algorithmus zum Berechnen von diskreten Logarithmen ist, der für jede beliebige Gruppe funktioniert. Der Silver-Pohlig-Hellmann Algorithmus funktioniert zwar auch für generische Gruppen, bietet jedoch nur bei Gruppen, deren Ordnung nicht prim ist, entscheidende Vorteile.

5.4 Diffie-Hellman in generischen Gruppen

Analog zum Diskreten Logarithmus Problem soll an dieser Stelle das Diffie-Hellman Problem in generischen Gruppen betrachtet werden.

Das Orakel und das Simulations-Orakel funktionieren genauso wie es für diskrete Logarithmen bereits gezeigt wurde. Der einzige Unterschied besteht darin, dass hier zwei Werte $x, y \in \mathbb{Z}_p$ gewählt werden und der Algorithmus A zum Lösen des Diffie-Hellman Problems

$$\sigma(0), \sigma(1), \sigma(x), \sigma(y)$$

erhält und $\sigma(xy)$ bestimmen muss.

Das Simulations-Orakel rechnet diesmal mit Polynomen aus $\mathbb{Z}_p[X, Y]$.

Auch hier gibt es wieder zwei Ereignisse, die den Erfolg des Algorithmus A definieren:

(E1) Der Algorithmus A hat ein Polynom P in der internen Liste des Orakels erzeugt, so dass

$$P(x, y) = xy$$

(E2) Der Algorithmus A kann zwischen Original- und Simulations-Orakel unterscheiden

Um Aussagen über die Erfolgswahrscheinlichkeit von A treffen zu können, benötigen wir folgenden Satz:

5.3 Lemma (Lemma von Schwarz)**.** *Sei $P \in \mathbb{Z}[X_1, \ldots, X_n]$, dann hat P höchstens*

$$\alpha p^{n-1}$$

Nullstellen, wobei d der Grad von P ist. (Nullstellen sind hier n-Tupel.)

Betrachtungen zu E1

Die Betrachtungen werden analog zum Szenario mit diskreten Logarithmen durchgeführt. Es gilt

$$P(x, y) = xy \qquad (P - XY)(x, y) = 0$$

Da P nur linear oder konstant sein kann, ergibt sich die Wahrscheinlichkeit zu

$$\text{Prob}\left((P - XY)(x, y) = 0\right) \leqslant \frac{2p}{p^2} = \frac{2}{p}$$

Damit folgt für die Wahrscheinlichkeit für E1

$$\text{Prob}(E1) \leqslant \frac{2m}{p}$$

Betrachtungen zu E2

Auch hier geht man analog zum Szenario mit diskreten Logarithmen vor.

Alle Polynome sind entweder linear oder konstant. Daher ergibt sich unter Anwendung des Lemma 5.3

$$\text{Prob}(E2) \leqslant \frac{m(m-1)}{p}$$

Für die Gesamt-Erfolgswahrscheinlichkeit zum Berechnen des Diffie-Hellman Problems mit höchstens m Gruppenoperationen gilt somit

$$\leqslant \frac{2}{p} + \frac{m(m-1)}{p}$$

Kapitel 6

Pairing-Based Cryptography

6.1 Bilineare Abbildungen

Ein wesentlicher Bestandteil von Pairings sind bilineare Abbildungen. G_1 und G_2 seien Gruppen mit der primen Ordnung p. G_1 sei eine additive und G_2 eine multiplikative Gruppe. P und Q seien Generatoren in G_1. Ein Pairing ist eine Abbildung

$$e : G_1 \times G_1 \to G_2$$

mit folgenden Eigenschaften:

1. Bilinearität: Für alle $P, Q \in G_1$ und für alle $a, b \in \mathbb{Z}_p^*$ gilt:

$$e(aP, bQ) = [e(P, Q)]^{ab}$$

2. Nicht-Degenerierbarkeit: Für alle $P \in G_1,\ P \neq 0$ gilt:

$$e(P, P) \neq 1$$

3. Effiziente Berechenbarkeit

6.1 Beispiel. *$G_1 = \mathbb{F}_2^n$ und $G_2 = \mathbb{F}_2$*

$$\begin{aligned} e : G_1 \times G_1 &\to G_2 \\ (a, b) &\mapsto < a, b > \end{aligned}$$

Diese Abbildung ist bilinear.

6.2 Beispiel. *$G_1 = (\mathbb{Z}_{p-1}, +), G_2 = (\mathbb{Z}_p^*, \cdot)$ und g sei ein Erzeuger von G_2*

$$e : (a, b) \mapsto g^{ab}$$

Diese Abbildung ist bilinear, nicht-degeneriert und effizient berechnenbar, aber für unsere Anwendung nutzlos, da der diskrete Logarithmus (DL) in G_1 leicht zu berechnen ist.

6.2 Eigenschaften vom DL für Gruppen mit Pairings

6.3 Theorem. *Das diskrete Logarithmus-Problem in G_1 ist nicht schwerer als das diskrete Logarithmus-Problem in G_2.*

Beweis: Seien $P, Q \in G_1$ gegeben ($P \neq 0$). Gesucht ist $\log_P Q$, d.h. gesucht ist ein $a \in \mathbb{N}$, so dass

$$aP = Q$$

Berechne

$$e(P, P) = g_1$$

und

$$e(P, Q) = g_2$$

Dann gilt

$$g_2 = e(P, Q) = e(P, aP) = e(P, P)^a = g_1^a$$

D.h. wenn man den DL in G_2 berechnen kann, kann man den DL auch in G_1 berechnen. □
Die Umkehrung ist (hoffentlich) falsch, wie man in Beispiel 6.2 sieht.

6.3 Decisional Diffie Hellman-Problem (DDH-Problem)

Gegeben seien $g, g^a, g^b \in G$ und zwei Elemente g^{ab}, g^r mit $a, b, r \in_R \mathbb{N}$ in zufälliger Reihenfolge.

Problem: Finde die Reihenfolge der beiden Elemente, d.h unterscheide g^{ab} und g^r.

Bemerkung: Wenn das DH-Problem leicht zu lösen ist, dann auch das DDH-Problem. Hierzu berechnet man einfach g^{ab} und testet, ob dies das erste oder zweite Element ist.

6.4 Theorem. *In der Gruppe G_1 ist das DDH-Problem leicht zu lösen.*

Beweis: siehe Übungsblatt 10

Bemerkung: In der Praxis werden für G_1 Elliptische Kurven und für G_2 Untergruppen der multiplikativen Gruppe von endlichen Körpern benutzt. Das Pairing ist dann entweder das Weil- oder das Tate-Pairing bzw. Varianten davon. Wir benötigen hier nur die abstrakten Eigenschaften.

6.3.1 Anwendungen

3-Parteien, 1-Runde-Schlüsselaustausch (DH für 3 Parteien) Die Parteien A, B, C besitzen jeweils einen geheimen Schlüssel $a, b, c \in \mathbb{Z}_p^*$ und einen öffentlichen Schlüssel $aP, bP, cP \in G_1$. Die Partei A sendet ihren öffentlichen Schlüssel aP an die Parteien B und C. Ebenso sendet B bP an A, C und C cP an A, B. Daraufhin berechnet A

$$e(bP, cP)^a = e(P, P)^{abc}$$

B berechnet

$$e(aP, cP)^b = e(P, P)^{abc}$$

und C berechnet

$$e(aP, bP)^c = e(P, P)^{abc}$$

Der gemeinsame Schlüssel ist also $e(P, P)^{abc}$
Die Sicherheit dieses Protokolls beruht auf folgendem Bilinear-DH-Problem (BDH-Problem): Bei gegebenen aP, bP, cP ist es praktisch unmöglich, $e(P, P)^{abc}$ auszurechnen.
Wenn das DH-Problem in G_1 leicht zu lösen ist, dann auch das BDH-Problem. Dazu berechnet man zunächst abP und anschließend

$$e(abP, cP) = e(P, P)^{abc}$$

Identity-Based Cryptography Im Jahr 1984 hat Shamir vorgeschlagen ein Kryptosystem zu entwickeln, in welchem die öffentlichen Schlüssel der Personen z.B. die Mail-Adressen sein können. Nach Erhalt einer verschlüsselten Nachricht kann der zugehörige geheime Schlüssel von einer vertrauenswürdigen Partei bezogen werden. Im Folgenden wird das Verfahren vorgestellt.
Die Gruppen G_1 und G_2 sowie die Abbildung e seien wie oben definiert. Weiterhin gibt es einen systemweiten geheimen Schlüssel $s \in \mathbb{N}$. Diesen kennt nur die vertrauenswürdige Instanz. Der öffentliche Schlüssel $Pk \in G_1$ sei sP mit $P \in G_1$.

Verschlüsselung: Es wird eine Nachricht m für A verschlüsselt. A ist z.B. eine Mail-Adresse.

$$E(Pk, A, m) = (rP, m \oplus H_2(g_A^r))$$

wobei $r \in_R \mathbb{N}, g_A = e(Q_A, Pk)$ und $Q_A = H_1(A)$.
H_1 und H_2 seien random oracle (z.B. Hash-Funktionen):

$$H_1 : [0,1]^* \to G_1$$

$$H_2 : G_2 \to [0,1]^*$$

Entschlüsselung: A fragt bei der vertrauenswürdigen Instanz nach dem privaten Schlüssel. Dieser ist $sQ_A = H_1(A)$. A berechnet

$$e(sQ_A, rP) = e(Q_A, P)^{sr} = [[e(Q_A, P)]^s]^r = [e(Q_A, sP)]^r = [e(Q_A, Pk)]^r = g_A^r$$

Damit kann A $H_2(g_A^r)$ berechnen und somit die Nachricht m entschlüsseln.

6.4 Signieren mit Pairings

Gegeben ist wieder die obige bilineare Abbildung

$$e : G_1 \times G_1 \to G_2$$

mit $|G_1| = |G_2| = p$ und p prim.
Eine Nachricht $m \in \{0,1\}^*$ soll signiert werden.

Annahme: Gegeben ist eine Hashfunktion $\{0,1\}^* \to G_1$, welche Bitstrings auf Punkte einer elliptischen Kurve abbildet.

Schlüssel-Generierung: $x \in_R \mathbb{Z}_p^*$ sei der geheime Schlüssel (Sk). Der öffentliche Schlüssel (Pk) sei $Q = xP$, wobei P ein öffentlicher Parameter ist $(P \neq 0)$.

Signierung: Es wird $H(m) \in G_1$ berechnet. Die Signatur der Nachricht m ist

$$\sigma = xH(m)$$

Verifikation: Die beiden Pairings $e(P, \sigma)$ und $e(Q, H(m))$ werden berechnet. Ist

$$e(P, \sigma) = e(Q, H(m))$$

dann ist die Signatur gültig. Stimmen die beiden Pairings nicht überein, ist die Signatur ungültig.

Überprüfung der Korrektheit: Wenn σ eine gültige Signatur für m ist, dann gilt

$$e(P, \sigma) = e(P, xH(m)) = e(P, H(m))^x = e(xP, H(m)) = e(Q, H(m))$$

Um eine Signatur einer Nachricht m zu erzeugen, muß ein Angreifer aus P, Q und $H(m)$ eine Signatur berechnen. Das ist äquivalent zum Lösen des DH-Problems:

$$(P, Q, H(m)) = (P, xP, aP)$$

Es gibt ein $a \in \mathbb{Z}_p^*$, so dass $aP = H(m)$. Gesucht ist

$$xH(m) = (xa)P$$

Vorteile der Signatur mittels Pairings:

- Die Signaturen sind sehr kurz, da Element in G_1. Ist G_1 Untergruppe einer Elliptischen Kurve, gilt z.B. $\log_2 \sigma \approx 160$ für ein mit $RSA - 1024$ vergleichbares Sicherheitslevel. Im Fall von DSA geht man von 320 Bit Signaturen aus.
- Das Berechnen der Signatur ist sehr effizient

Nachteil: Die Verifikation einer Signatur ist relativ langsam, da das Berechnen des Pairings sehr zeitaufwendig ist.

Abstraktion: Obiges Signatur-Schema funktioniert für alle Gruppen, in denen das DH-Problem schwer und das DDH-Problem leicht zu lösen ist. Diese Gruppen nennt man auch GAP-Gruppen.

6.4.1 Mehrfachsignaturen

Bei dieser Variante erstellen n Parteien für eine Nachricht m eine gemeinsame Signatur. Jede dieser Parteien besitzt einen eigenen geheimen Schlüssel $x_i \in \mathbb{Z}_p^*$ sowie den zugehörigen öffentlichen Schlüssel (Pk) $Q_i = x_i P \in G_1$ und berechnet die jeweilige Signatur

$$\sigma_i = x_i H(m)$$

Die gemeinsame Signatur $\sigma = \sum \sigma_i$ ist eine Signatur für die Nachricht m und den öffentlichen Schlüssel $Q = \sum Q_i$

Überprüfung der Korrektheit: Es gilt

$$\begin{aligned} e(P, \sigma) &= e(P, \sum \sigma_i) = e(P, H(m) \sum x_i) \\ &= e(P, H(m))^{\sum x_i} = e(P \sum x_i, H(m)) \\ &= e(Q, H(m)) \end{aligned}$$

6.4.2 Threshold-Signaturen

Idee: Es gibt n Parteien und einen gemeinsamen Pk. Mehr als $(t-1)$ Parteien sollen in der Lage sein, Signaturen für Nachrichten m und den gemeinsamen Pk zu erzeugen $(t \leqslant n)$.

Einschub: Shamirs Secret Sharing (n, t) Ein Geheimnis $s \in \mathbb{F}_p$ wird auf n Parteien aufgteilt, so dass

- mehr als $(t-1)$ Parteien das Geheimnis s rekonstruieren können.
- weniger als t Parteien keinerlei Informationen über s erhalten.

Um ein Geheimnis $s \in \mathbb{F}_p$ aufzuteilen, werden zufällig t Koeffizienten $f_i \in \mathbb{F}_p$ gewählt und für $i \in \{1, \ldots, t\}$

$$f(x) = (\sum_{i=1}^{t} f_i x^i) + s$$

berechnet. Das Geheimnis s wird also durch $f(0)$ dargestellt. Das Teil-Geheimnis für die Partei i mit $i \in \{1, \ldots, n\}$ ist $x_i = f(i)$. Da mit t Punkten (i, x_i) das Polynom f vom

Grad $Grad(f) \leqslant t$ eindeutig bestimmt ist, können t Parteien (oder mehr) das Geheimnis s rekonstruieren. Dazu wird z.B. die Lagrange-Interpolation verwendet, mit der man für gegebene Werte $(i, x_i), i \in S \subseteq \{1, \ldots, n\}, |S| \geqslant t$

$$f(z) = \sum_{i \in S} x_i L_i(z)$$

berechnet, wobei

$$L_i(z) = \prod_{\substack{j \in S \\ j \neq i}} \frac{z-j}{i-j}$$

Es gilt für $z \in S$

$$L_i(z) = \begin{cases} 1, & \text{wenn } z = i \\ 0, & \text{wenn } z \neq i \end{cases}$$

denn

$$L_i(i) = \prod_{\substack{j \in S \\ j \neq i}} \frac{i-j}{i-j} = 1$$

und

$$L_i(z) = \prod_{\substack{j \in S \\ j \neq i}} \frac{z-j}{i-j} = \left(\prod_{\substack{j \in S \\ j \neq i \\ j \neq z}} \frac{z-j}{i-j} \right) \left(\frac{j-j}{i-j} \right) = 0$$

Es gilt

$$f(0) = \sum_{i \in S} L_i(0) x_i$$

d.h. das Geheimnis $s = f(0)$ ist eine Linearkombination der Teilgeheimnisse, wobei die Koeffizienten $L_i(0)$ nur von der Teilmenge $S \subseteq \{1, \ldots, n\}$ abhängen.

Zurück zur Threshold-Signatur: Jede Partei wählt einen geheimen Schlüssel $x_i \in \mathbb{Z}_p$ und veröffentlicht $Q_i = x_i P$. Der gemeinsame öffentliche Schlüssel ist

$$Q = \sum_{i \in S} L_i(0) Q_i$$

für $|S| \geqslant t$.
Um eine Nachricht m zu signieren, berechnet jede Partei aus S eine Signatur

$$\sigma_i = x_i H(m)$$

Die gemeinsame Signatur für m und den öffentlichen Schlüssel Q ist

$$\sigma = \sum_{i \in S} L_i(0) \sigma_i$$

Überprüfung der Korrektheit: Es gilt

$$\begin{aligned} e(P,\sigma) &= e(P,\sum L_i(0)\sigma_i) = e(P,H(m)\sum L_i(0)x_i) \\ &= e(P,H(m))^{\sum L_i(0)x_i} = e(P\sum L_i(0)x_i, H(m)) \\ &= e(Q_i\sum L_i(0), H(m)) = e(Q,H(m)) \end{aligned}$$

6.4.3 Gesammelte Signaturen

Eine Nachricht m_i wird mit dem öffentlichen Schlüssel Q_i signiert, $i \in \{1,\ldots,n\}$.

Idee: Überprüfe alle Signaturen auf einmal.
Setze

$$\sigma = \sum \sigma_i = \sum x_i H(m_i)$$

Teste, ob

$$e(P,\sigma) = \prod e(Q_i, H(m_i))$$

D.h. es gibt eine gemeinsame Signatur für alle Nachrichten m_i.

Algorithmenverzeichnis

Literaturverzeichnis

[1] Johannes Buchmann, *Einführung in die Kryptographie*, Springer, 2003

[2] Peter Bundschuh, *Einführung in die Zahlentheorie*, Springer, 2002

[3] Michael J. Wiener, *Cryptanalysis of Short RSA Secret Exponents*, 1989

[4] Otto Forster, *Algorithmische Zahlentheorie*, Vieweg Verlagsgesellschaft, 1996

[5] Neal Koblitz, *A Course in Number Theory and Cryptography*, 2nd Edition, Springer, 1994

[6] Francois Morain, Jorge Olivos, *Speeding up the computations on an elliptic curve using addition-subtraction chains*, ftp://ftp.inria.fr/INRIA/publication/Theses/TU-0144/ch4.ps, 4. Kapitel der Dissertation von F. Morain)

[7] Ran Canetti, Ron Rivest, *Special Topics in Cryptography*, 2004