RSA und kleine modulare Wurzeln

Angriff auf RSA mit kleinem Exponenten

Im vorigen Semester hatten wir Angriffe auf RSA behandelt, darunter den Broadcast Angriff von Hastad, welcher auf einem Satz von Coppersmith zur Berechnung kleiner Wurzeln von modularen Gleichungen beruht. Der Satz von Coppersmith beruht seinerseits auf Gittertechniken und wir hatten den Beweis damals ausgelassen. Das soll jetzt nachgeholt werden.

Es sei zunächst an den Satz von Coppersmith und den Satz von Hastad erinnert.

1 Satz (Coppersmith)**.** *Sei $f \in \mathbb{Z}[t]$ normiert und $s = \deg(f)$. Dann können alle Lösungen $f(m) = 0 \bmod n$ mit $m \in \mathbb{Z}$ und $|m| < n^{1/s}$ effizient aus f und n berechnet werden.*

2 Satz (Hastad)**.** *Seien n_i teilerfremd, $g_i \in \mathbb{Z}[x]$ normiert für $1 \le i \le k$ und $s = \max_i \deg(g_i)$. Es gebe ein eindeutiges $m < \min_i n_i$ mit $g_i(m) = 0 \bmod n_i$. Ist $k \ge s$, so kann m effizient aus den n_i und g_i berechnet werden.*

Beweis. Sei $n = \prod_i n_i$ und $g = \sum_i \lambda_i x^{s-\deg(g_i)} g_i$, wobei $\lambda_i = \delta_{i,j} \bmod n_j$, und $\sum_i \lambda_i = 1$. Solche λ_i existieren nach dem chinesischen Restsatz, da die n_j teilerfremd sind. Dann ist g normiert, $\deg(g) = s$ und $g(m) = 0 \bmod n$. Wegen $m < \min_i n_i \le n^{1/k} \le n^{1/s}$ kann m mit dem Satz von Coppersmith effizient bestimmt werden. $\square$

Effizient heißt hier wie üblich in Polynomzeit in der Länge der Eingabe. Der Satz von Hastad kann mit $g_i(m) = f_i(m)^{e_i} - c_i \bmod n_i$ und speziell $e_i = e$ und linearen f_i (z.B. festes Padding, $f_i(m) = m + b_i$) angewendet werden. Ein Angreifer, welcher bei kleinem e mindestens e, an verschiedene Personen gerichtete Verschlüsselungen $g_i(m)$ derselben Nachricht erhält, kann leicht m ausrechnen.

Das Polynom f kann exponentiell viele Nullstellen modulo n haben, wenn n das Produkt von ausreichend vielen verschiedenen Primzahlen ist, unabhängig vom Grad des Polynoms (beispielsweise $f(x) = x^2 - 1$ modulo $p_1 \cdots p_r$, hat 2^r Nullstellen). Eine Folgerung aus dem Satz von Coppersmith ist, daß es allerdings nur polynomiell viele kleine Nullstellen von f modulo n geben kann (andernfalls ließen diese sich nicht effizient berechnen).

Beweis vom Satz von Coppersmith

Sei $b \in \mathbb{R}^{>0}$ eine obere Schranke von $|m|$ und $h \in \mathbb{Z}^{\geq 1}$. Das Ziel ist es, diese Werte so zu bestimmen, daß es ein $g = \sum_i c_i x^i \in \mathbb{Z}[x]$ gibt mit

1. $g(m) = 0 \bmod n^{h-1}$,
2. $|g(m)| < n^{h-1}$.

Dann gilt nämlich $g(m) = 0$ und der Wert m kann als Nullstelle eines Polynoms über $\mathbb{Z}$ in Polynomzeit in $\deg(g)$ und $\max_i \log(|c_i|)$ berechnet werden. Hierfür kann man die Nullstellen als reelle Zahlen mit ausreichend großer Präzision ausrechnen und dann testen, oder man wendet Polynomfaktorisierung über $\mathbb{Z}$ an, welche ebenfalls auf dem LLL beruht. Das Polynom g enthält also alle Lösungen m mit $f(m) = 0 \bmod n$ als Nullstellen. Es gibt daher auch nur maximal $\deg(g)$ viele Lösungen.

Um das Polynom g zu finden, gehen wir wie folgt vor. Für $0 \leq i \leq h-1$ und $0 \leq j \leq s-1$ sei $g_{i,j}(x) = n^{h-i-1} x^j f(x)^i \in \mathbb{Z}[x]$ und

$$\Lambda = \sum_{0 \leq i \leq h-1,\, 0 \leq j \leq s-1} \mathbb{Z}\, g_{i,j}(x).$$

Die Grade $\deg(g_{i,j}) = si + j$ sind alle verschieden. Die $g_{i,j}$ bilden daher eine Basis von Λ und Λ ist ein freier $\mathbb{Z}$-Modul vom Rang sh. Für jedes $h \in \Lambda$ gilt die Bedingung 1, $h(m) = 0 \bmod n^{h-1}$, und wir suchen nach g in Λ. Um auch Bedingung 2 zu erreichen, führen wir eine Norm auf Λ ein und machen es so zu einem Gitter. Für $h = \sum_i d_i x^i \in \Lambda$ definieren wir $||h||_{1,b} = \sum_i |d_i b^i|$ und $||h||_{2,b} = (\sum_i (c_i b^i)^2)^{1/2}$. Dann ist Λ zusammen mit $|| \cdot ||_{2,b}$ ein Gitter, da die Norm offensichtlich von einem Skalarprodukt herkommt. Man kann Λ isometrisch in $\mathbb{Z}^{sh}$ mit dem euklidischen Skalarprodukt einbetten, indem man die Polynome als Vektoren schreibt und dann x durch b ersetzt. Es gilt $|g(m)| \leq ||g(x)||_{1,b} < (sh)^{1/2} ||g(x)||_{2,b}$. Wir wollen g so wählen, daß wir $(sh)^{1/2} ||g(x)||_{2,b} \leq n^{h-1}$ erreichen.

Wegen der Diagonalgestalt der $g_{i,j}$ gilt

$$\begin{aligned} d(\Lambda) &= \prod_{i,j} n^{h-i-1} b^{si+j} = n^{\sum_{i,j}(h-i-1)} \prod_{i,j} b^{si+j} \\ &= n^{s\sum_i (h-i-1)} b^{sh(sh-1)/2} = n^{sh(h-1)/2} b^{sh(sh-1)/2}. \end{aligned}$$

Der LLL Algorithmus mit $\delta = 3/4$ berechnet einen kurzen Vektor $g \in \Lambda$ der Länge

$$||g||_{2,b} \leq 2^{(sh-1)/4} d(\Lambda)^{1/(sh)}.$$

Daraus erhalten wir die hinreichende Bedingung

$$(sh)^{1/2} 2^{(sh-1)/4} n^{(h-1)/2} b^{(sh-1)/2} \leq n^{h-1}.$$

Wählen wir b so, daß diese Bedingung erfüllt ist, dann gilt neben der Bedingung 1 auch die Bedingungen 2 oben für g. Da wir an einem maximalen b interessiert sind, ergibt sich

$$\begin{aligned} b &= (n^{(h-1)/2} / ((sh)^{1/2} 2^{(sh-1)/4}))^{2/(sh-1)} \\ &= n^{(h-1)/(sh-1)} / (2^{1/2} (sh)^{1/(sh-1)}). \end{aligned}$$

Der Ausdruck auf der rechten Seite ist monoton wachsend in h und konvergiert gegen $n^{1/s}/2^{1/2}$. Wir wollen h jedoch nur so groß wählen, daß das Verfahren effizient bleibt. Für $sh \geq 2$ erhalten wir die folgende Abschätzung

$$\begin{aligned} b &= n^{(h-1)/(sh-1)} / (2^{1/2} (sh)^{1/(sh-1)}) \\ &\geq n^{(h-1)/(sh)} / 2^{3/2} \\ &\geq n^{(1-1/h)/s} / 2^{3/2} \end{aligned}$$

Für $h = \lceil \max\{\log(n), 2/s\} \rceil$ gilt $n^{1-1/h} \geq \exp(\log(n) - 1) = n/e$, folglich ergibt sich

$$b \geq n^{1/s} / (e 2^{3/2}).$$

Damit ist der Satz für $n^{1/s}/c$ und einer absoluten Konstanten $c > 1$ richtig. Er ist es aber auch für $n^{1/s}$ wie folgt. Sei $t = \lfloor n^{1/s}/c \rfloor$. Für $t = 0$ ist nichts zu testen (keine Lösungen). Für $t > 0$ schreiben wir $m = dt + m_0$ mit $0 \leq m_0 < t$. Es gilt $|d| \leq \lceil n^{1/s}/t \rceil \leq 2c + 1$. Das Problem $f(m) = 0 \bmod n$ wird damit zu maximal $4c + 2$ Problemen $f_d(m_0) = 0 \bmod n$ mit $f_d(x) = f(dt + x)$ und $|m_0| \leq n^{1/s}/c$, und diese können mit obiger Methode effizient gelöst werden.

Bemerkungen

Der Satz von Coppersmith hat eine Reihe weiterer, kryptographierelevanter Anwendungen. Hier ist sind ein paar:

- Geheimen Schlüssel d in RSA ausrechnen, wenn die unteren $\log_2(d)/4$ Bits bekannt sind.

- Geheimen Schlüssel d in RSA ausrechnen, wenn $d \leq n^{0.292}$ (mit Variante vom Satz von Coppersmith).

- Faktorisieren von $n = pq$, wenn die Hälfte der unteren oder oberen Bits von p bekannt sind.

- Faktorisieren von $n = p^r q$, wenn r groß ist.

Der Satz von Coppersmith kann auf multivariate Polynome verallgemeinert werden, allerdings nur heuristisch. Will man beispielsweise $f(m_1, m_2) = 0 \bmod n$ lösen, so bekommt man durch eine ähnliche Gittertechnik ein Polynom $g(x, y)$ mit $g(m_1, m_2) = 0$ heraus. Die Bestimmung solcher Lösungen ist aber im allgemeinen schwer (Stichwort „Ganze Punkte auf Kurven"). Daher versucht man im Gitter zwei solche Polynome g_1, g_2 zu finden, und betrachet dann die Resultante $\mathrm{res}_y(g_1, g_2) \in \mathbb{Z}[x]$, deren Nullstellen die m_1 sind. Das (theoretische) Problem hierbei ist, daß die Resultante auch Null sein kann. Dies ist zum Beispiel grundsätzlich der Fall für $f(x, y) = x - y$, da dieses Polynom zu viele Nullstellen hat, als daß eine multivariate Version vom Satz von Coppersmith gelten könnte.

Ein weiterer Satz von Coppersmith sagt, daß man alle kleinen Werte m_1, m_2 mit $f(m_1, m_2) = 0$ effizient aus f ausrechnen kann. Für multivariate Polynome gilt dies nur noch heuristisch.