Seminar IT-Sicherheit

Cryptanalytic Attacks on RSA

Discrete Logarithm Attacks

von

Michael Greßmann

(winf100373@fh-wedel.de)

im

Sommersemester 2016

Dozent:

Prof. Dr. Gerd Beuster

(gb@fh-wedel.de)

Eingereicht am 05.06.2016

Inhaltsverzeichnis

1. Einleitung

Seitdem das von Ronald L. Rivest, Adi Shamir und Leonard Adleman im Jahr 1977 entwickelte asymmetrische Verschlüsselungsverfahren „RSA“ veröffentlicht wurde, avancierte es bis zur heutigen Zeit zum bedeutendsten seiner Art. Insbesondere im Hinblick auf unsere weltweit vernetzte Kommunikation trägt dieses Verfahren maßgeblich dazu bei, Datenaustausch sicher zu betreiben; sei es E-Mail (OpenPGP, S/MIME), Electronic Banking (HBCI) oder Übertragungsprotokolle (TLS, SSH). Sobald sensibler Datenverkehr stattfindet, ist RSA nicht wegzudenken. All diese Anwendungsbereiche sind auf die Sicherheit dieses Verfahrens angewiesen. Die Tatsache, dass unter Beachtung der korrekten Anwendung (mindestens RSA-2048) noch keine Entschlüsselung möglich war, zeigt die Robustheit, auf der dieser Algorithmus basiert. Jedoch sind seit der Veröffentlichung des Verfahrens, dessen Ziel es eigentlich war, die Theorie zur Public-Key-Kryptographie von Whitfield Diffie und Martin Hellman zu widerlegen, immer wieder Versuche unternommen worden, um RSA zu brechen. Solange diese nicht zum Erfolg führen und damit die Primfaktorzerlegung nicht in Polynomialzeit bestimmt werden kann, bleibt auch die Entschlüsselung von RSA praktisch nicht lösbar.

In dem Buch „Cryptanalytic Attacks on RSA“ von Song Y. Yang [1] werden alle bisher bekannten kryptoanalytischen Angriffe und Abwehrmaßnahmen auf beziehungsweise von RSA aufgezeigt. Dabei ist zu erwähnen, dass das Discrete Logarithm Problem (DLP) hinreichend ähnlich zum Integer Factorization Problem (IFP) ist (siehe Kapitel 2.1 sowie 2.2) und somit Lösungsansätze für beide Probleme verwendet werden können. Diese Seminararbeit beschäftigt sich mit dem Kapitel 4 (Discrete Logarithm Attacks) des Buchs von Yang [1, S. 111 – 133] und beschreibt Algorithmen, die das Potential besitzen, das DLP in polynomieller Zeit zu lösen.

2. Der diskrete Logarithmus

2.1. Das diskrete Logarithmus Problem

In der Zahlen- und Gruppentheorie stellt der diskrete Logarithmus das Pendant zum klassischen Logarithmus der Analysis. Jedoch ist der Zahlenbereich nicht reell, sondern diskret. Bekanntermaßen ist die Umkehrfunktion des diskreten Logarithmus die diskrete Exponentialfunktion. Sie ist selbst für große Exponenten effizient berechenbar. Wird der Logarithmus einer ganzen Zahl zur Basis einer ganzen Zahl modulo einer Primzahl gesucht, spricht man vom diskreten Logarithmus Problem (DLP):

$$\text{DLP:} \quad \{n \in \mathbb{Z}^+_{>1}, x, y, k \in \mathbb{Z}^+, y \equiv x^k \pmod{n}\} \xrightarrow{\text{suche}} \{k\}. \text{ [1, S. 111]}$$

Für dieses Problem existiert bisher kein Algorithmus, der die Lösung effizient im Sinne der Komplexitätstheorie bestimmt. Diese Eigenschaft der Einwegfunktionen macht sich nicht nur das RSA-Kryptosystem zu Nutze. Andere bekannte Kryptosysteme, wie der Diffie-Hellman-Merkle Schlüsselaustausch, das Elgamal-Signaturverfahren und der Digital Signature Standard/Algorithm (DSS/DSA), basieren ebenfalls darauf, dass der Rechenaufwand zum Lösen bei entsprechend groß gewählten Zahlen (bspw. RSA-2048) bisher nicht in polynomieller Zeit erfolgen kann. [1, S. 111]

Die Kryptographie zieht einen praktischen Nutzen aus einem Spezialfall der zyklischen Gruppen, indem diese ein Element der primen Restklassengruppen ist (modulo Primzahlen). Das folgende rudimentäre Beispiel gibt einen Eindruck dessen, wie schwer das DLP für die in der Realität auftretenden Schlüsselgrößen von 2048-Bit zu lösen ist. Gegeben sei eine multiplikative Gruppe $(\mathbb{Z}^*_p, x_p)$ aus $L = \{1, \ldots, p-1\}$ modulo p, für $p = 7$:

x_7	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Gesucht sei der kleinste Wert für $k \in L$, sodass $3^k = 4$.

Lösung: $3 \; x_7 \; 3 = 2$

$2 \; x_7 \; 3 = 6$

$6 \; x_7 \; 3 = 4$

Daraus folgt, dass $3^4 \equiv 4 \pmod 7$, was durch $\log_3 4 \equiv 4 \pmod 7$ belegt wird. Daher ist es klar, dass bei entsprechend groß gewählten Zahlen ein derartiges Vorgehen ineffizient ist.

Ein Angriff auf dieses Problem stellt somit den Versuch dar, alle auf dem DLP aufbauenden Kryptosysteme zu brechen. Gleichzeitig fallen darunter auch die Systeme, die auf dem IFP basieren, da diese Probleme eng miteinander verbunden sind [1, S 111]. Mit einem effizienten Integer Factorization Algorithmus ist es dem Angreifer möglich, $N = pq$ zu faktorisieren und daher auch $d \equiv 1/e \bmod ((p-1)(q-1))$ zu berechnen. Mit diesen Informationen lassen sich dann ebenfalls die verschlüsselten Daten nach $C \equiv M^d \pmod N$ wiederherstellen. Steht dem Angreifer ein effizienter Discrete Logarithm Algorithmus zur Verfügung, kann dieser seinen eigenen Klartext M erstellen und die öffentlichen Schlüsselinformationen (e, N) dazu nutzen, um daraus den korrespondierenden Geheimtext C zu M zu bekommen. Anschließend benutzt er den Discrete Logarithm Algorithmus, um $d \equiv \log_M{}^e \; M \pmod N$ zu berechnen. Das bedeutet im gleichen Zug, dass man nicht nur das DLP, sondern auch das RSA System in polynomieller Zeit lösen beziehungsweise brechen kann.

2.2. Beispiel

Im dieser Arbeit zugrunde liegenden Buch [1] wird Kapitel 4.1.2 (S. 112 f.) ein Beispiel vorgestellt, bei dem es nicht Ziel ist, N zu faktorisieren, sondern einen alternativen Lösungsweg einzuschlagen, sodass der Geheimtext C rekonstruiert und somit ein korrespondierendes d gefunden wird.

Gegeben seien der Public-Key und der Cipher-Text wie folgt:

$$(e, N, C) = (7, 69056069, 19407420)$$

Gesucht sei der Klartext M, sodass $C \equiv M^e \pmod{N}$. Offensichtlicher weise muss folgende Berechnung ausgeführt werden, um d zu finden und die Verschlüsselung zu invertieren:

$$d \equiv 1/e \pmod{\phi(N)}, \text{ sodass } M \equiv C^d \pmod{N}$$

Um dieses jedoch durchzuführen, muss man N faktorisieren. Dies soll, wie bereits angekündigt, nicht die angestrebte Intention sein. Es wird ein anderer Weg propagiert, der folgendermaßen lautet:

(1) Man wählt einen zufälligen Klartext M, für den gelten muss, dass $\text{ggT}(N, M) = 1$, also teilerfremd zu N ist. Es wird $M = 59135721$ gewählt und folgende Berechnung ausgeführt:

$$C \equiv M^e \equiv 59135721^7 \equiv 60711351 \pmod{69056069}.$$

(2) Berechne d, indem der diskrete Logarithmus M zur Basis C modulo N genommen wird:

$$d \equiv \log_C M \equiv \log_{60711351} 59135721 \equiv 4931383 \pmod{69056069}.$$

(3) Auch lässt sich e berechnen, indem der diskrete Logarithmus C zur Basis M modulo N genommen wird:

$$e \equiv \log_M C \equiv \log_{59135721} 60711351 \equiv 7 \pmod{69056069}.$$

(4) Nun kann der gegebene Cipher-Text $C = 19407420$ mit dem berechneten d bestimmt werden:

$$M \equiv C^d \equiv 19407420^{4931383} \equiv 7289258 \pmod{69056069}.$$

(5) Das Ergebnis kann sehr leicht verifiziert werden, da:

$$C \equiv M^e \equiv 7289258^7 \equiv 19407420 \pmod{69056069}.$$

Normalerweise ist ein Angreifer dazu gezwungen, $d \equiv 1/e \pmod{\phi(N)}$ zu berechnen, um C zu finden und $M \equiv C^d \pmod{N}$ schlussendlich herauszubekommen. Mit dem oben beschriebenen Beispiel wurde gezeigt, dass man d auf eine andere Weise bestimmen kann, indem von

$$d \equiv \log_{C'} M' \bmod N$$

Gebrauch gemacht wird, sodass M' und C' relativ einfach errechnet wurden. Die wichtigsten Erkenntnisse hierbei sind aber, dass das d aus M' und C' dazu verwendet werden kann, um C zu entschlüsseln und damit M zu gewinnen. Dabei muss allerdings angemerkt werden, dass das errechnete d aus $\log_{C'} M'$ mod N sich zwar vom d aus $1/e$ (mod $\phi(N)$) unterscheidet, aber sich dennoch identisch bei der Entschlüsselung von C verhält:

$$M \equiv 19407420^{4931383} \equiv 7289258 \text{ (mod } 69056069),$$

$$M \equiv 19407420^{39451063} \equiv 7289258 \text{ (mod } 69056069).$$

Das d aus $\log_{C'} M'$ mod N mag kleiner sein als das d aus $1/e$ (mod $\phi(N)$), aber diese wesentliche Information genügt, um den Verdacht aufkommen zu lassen, dass das RSA-Kryptosystem doch gebrochen werden kann, denn ein Angreifer kann immerhin viele verschiedene d's ausprobieren, die kleiner als der tatsächliche Wert sind.

2.3. Kategorisierung von DLP Algorithmen

Allgemein lassen sich laut Song Y. Yang [1, S. 114 f.] Algorithmen, die versuchen, das DLP zu lösen, in drei verschiedene Kategorien unterteilen:

(1) Algorithmen, die nicht versuchen, spezielle Eigenschaften von Gruppen auszunutzen und somit für willkürliche Gruppen anwendbar sind. Zu diesen zählen Daniel Shanks' Baby-Step Giant-Step Algorithmus [4], John M. Pollards ρ-Methode [2] und die λ-Methode [3].
(2) Algorithmen für endliche Gruppen, dessen Ordnung keine großen Primfaktoren enthalten; speziell Gruppen gleichmäßiger Ordnung. Ein Beispiel stellt der Silver-Pohlig-Hellman Algorithmus [6] dar.
(3) Die dritte Kategorie beinhaltet Algorithmen, die ausnutzen, dass Gruppenelemente als Produkte von Elementen aus einer relativ kleinen Menge repräsentiert werden können. Hier finden sich der Adlemans Index Calculus Algorithmus [7] und Gordons Number Field Sieve (NFS) [8] wieder.

Im Folgenden werden der Baby-Step Giant-Step, der Silver-Pohlig-Hellman Algorithmus, der Index Calculus und Gordons NFS dargestellt.

3. Diskrete Logarithmus Angriffe

3.1. Baby-Step Giant-Step Angriff

3.1.1. Algorithmus

Hält man sich das Beispiel aus 2.2 vor Augen, wird sofort deutlich, dass die Anzahl von Gruppenoperationen mit $O(n)$ ineffizient ist. Daher wird im Folgenden der Baby-Step Giant-Step Algorithmus von Daniel Shanks [4] vorgestellt, der auf einer Wurzelberechnung basiert und gezielter nach Lösungen von:

$x = \log_a b \bmod n$, wobei G eine endliche zyklische Gruppe der Ordnung n, a ein Erzeuger von G und $b \in G$ ist.

Diese Methode bedient sich der Beobachtung, dass man, wenn $x = \log_a b$, $x = i + jm$ schreiben kann, wobei $0 \leq i, j < m$ und $m = [\sqrt{n}]$. Eine Anwendung auf das Beispiel aus 2.2 ist also:

$4 = \log_3 4 \bmod 7$, sodass $a = 3$, $b = 4$, $m = 3$. Daraus ergibt sich $4 = 1 + 3 \cdot 1$.

Der genaue Algorithmus [1, S. 116] lautet wie folgt:

(1) Berechne $s = [\sqrt{n}]$.

(2) Berechne die erste Liste S (Baby-Step) mit den Paaren (ya^r, r), $r = 0, 1, 2, 3, \ldots, s - 1$:

$S = \{(y, 0), (ya, 1), (ya^2, 2), (ya^3, 3), \ldots, (ya^{s-1}, s - 1) \bmod n\}$

und sortiere S nach ya^r, dem ersten Element der Paare aus S.

(3) Berechne die zweite Liste T (Giant-Step) mit den Paaren (a^{ts}, ts), $t = 1, 2, 3, \ldots, s$:

$T = \{(a^s, 1s), (a^{2s}, 2s), (a^{3s}, 3s), \ldots, (a^{s^2}, s^2) \bmod n\}$

und sortiere T nach a^{ts}, dem ersten Element der Paare aus T.

(4) Suche in beiden Listen S und T nach einer Übereinstimmung $ya^r = a^{ts}$, wobei ya^r in S und a^{ts} in T liegt. Berechne anschließend $x = ts - r$. Der Wert von x ist das Gesuchte Ergebnis aus $\log_a y \pmod n$.

Dieser Algorithmus benötigt eine Tabelle mit $O(m)$ Einträgen ($m = [\sqrt{n}]$, bei dem n der Modulus entspricht). Unter Verwendung eines Sortieralgorithmus beläuft sich die Anzahl

der Gruppenoperationen auf $O(m \log m)$. Kombiniert mit der Berechnung eines diskreten Logarithmus ergibt sich schließlich eine Laufzeit von $O(\sqrt{n} \log n)$ und ein Speicherbedarf von $O(\sqrt{n})$ Gruppenelementen. Aber auch diese Methode stößt, bedingt durch groß genug gewählte Gruppenordnungen, an ihre Grenzen. Sobald diese die Größe von 10^{40} überschreitet, ist sie nicht mehr durchführbar.

3.1.2. Beispiele

Nachfolgend wird der unter 3.1.1 beschriebene Algorithmus verwendet, um den diskreten Logarithmus zu berechnen. Gegeben sei $x = \log_3 7 \bmod 19$, sodass $7 = 3^x \bmod 19$. Nun werden die Schritte nach Baby-Step Giant-Step durchgeführt:

(1) $y = 7$, $a = 3$ und $n = 19$, $s = [\sqrt{19}] = 4$.

(2) Berechnung des Baby-Steps:

$S = \{(y, 0), (ya, 1), (ya^2, 2), (ya^3, 3) \bmod 19\}$

$= \{(7, 0), (7 \cdot 3, 1), (7 \cdot 3^2, 2), (7 \cdot 3^3, 3) \bmod 19\}$

$= \{(7, 0), (2, 1), (6, 2), (18, 3)\}$

$= \{(2, 1), (6, 2), (7, 0), (18, 3)\}$

(3) Berechnung des Giant-Steps:

$T = \{(a^s, s), (a^{2s}, 2s), (a^{3s}, 3s), (a^{4s}, 4s) \bmod 19\}$

$= \{(3^4, 4), (3^8, 8), (3^{12}, 12), (3^{16}, 16) \bmod 19\}$

$= \{(5, 4), (6, 8), (11, 12), (17, 16)\}$.

(4) Übereinstimmung finden und x berechnen: In beiden Listen S und T kommt die Zahl 6 als erstes Element von Paaren vor, mit $r = 2$ und $st = 8$, sodass $x = st - r = 8 - 2 = 6$. Es bedeutet also: $\log_3 7 \pmod{19} = 6$ oder äquivalent dazu $3^6 \pmod{19} = 7$.

Für das zweite Beispiel gelte:

Gegeben sei $x = \log_{31} 47 \bmod 73$, sodass $47 = 31^x \bmod 73$. Nun werden die Schritte nach Baby-Step Giant-Step durchgeführt:

(1) $y = 47$, $a = 31$ und $n = 73$, $s = [\sqrt{79}] = 8$.

(2) Berechnung des Baby-Steps:

$S = \{(y, 0), (ya, 1), (ya^2, 2), (ya^3, 3) \bmod 73\}$

$= \{(47, 0), (47 \cdot 31, 1), (47 \cdot 31^2, 2), (47 \cdot 31^3, 3), (47 \cdot 31^4, 4), (47 \cdot 31^5, 5), (47 \cdot 31^6, 6), (47 \cdot 31^7, 7) \bmod 73\}$

$= \{(47, 0), (70, 1), (53, 2), (37, 3), (52, 4), (6, 5), (40, 6), (72, 7)\}$

$= \{(6, 5), (37, 3), (40, 6), (47, 0), (52, 4), (53, 2), (70, 1), (72, 7)\}$.

(3) Berechnung des Giant-Steps:

$T = \{(a^s, s), (a^{2s}, 2s), (a^{3s}, 3s), (a^{4s}, 4s), (a^{5s}, 5s), (a^{6s}, 6s), (a^{7s}, 7s), (a^{8s}, 8s) \bmod 73\}$

$= \{(31^8, 8), (31^{16}, 16), (31^{24}, 24), (31^{32}, 32), (31^{40}, 40), (31^{48}, 48), (31^{56}, 56), (31^{64}, 64) \bmod 73\}$

$= \{(4, 8), (16, 16), (64, 24), (37, 32), (2, 40), (8, 48), (32, 56), (55, 64)\}$

$= \{(2, 40), (4, 8), (8, 48), (16, 16), (32, 56), (37, 32), (55, 64), (64, 24)\}$.

Übereinstimmung finden und x berechnen: In beiden Listen S und T kommt die Zahl 37 als erstes Element von Paaren vor, mit $r = 3$ und $st = 32$, sodass $x = st - r = 32 - 3 = 29$. Es bedeutet also: $\log_{31} 47 \pmod{73} = 29$ oder äquivalent dazu $31^{29} \pmod{73} = 47$.

3.1.3. Gefahrenpotential für RSA

Die Eigenschaft der Reduktion des Suchbereichs der Übereinstimmung in beiden Listen auf $[\sqrt{n}]$ und die theoretische Laufzeit von $O(\sqrt{n} \log n)$ lässt den Eindruck einer echten Gefahr für das DLP und damit der Sicherheit von RSA erwecken. Der eigentliche Sinn hinter dem Baby-Step Giant-Step Algorithmus ist die Bestimmung der Ordnung eines Elements g in einer Gruppe G. Mit der dargelegten Verwendung wird aber dessen Idee dazu benutzt, um diskrete Logarithmen zu berechnen [1, S. 116]. Weiterhin ist keine Beschränkung auf gewisse Gruppen gegeben. Allerdings liegt in der Vorberechnung beider Listen auch gleichzeitig eine Problematik dieses Algorithmus. Mit Pollards ρ-Methode [2] und der λ-Methode [3] sind zwei weitere auf Wurzelberechnung basierende Algorithmen bekannt, die jenen Nachteil nicht aufweisen. Die probabilistische Natur beider Methoden

erfordert dadurch $O(n)$ Gruppenoperationen und wird mit Erreichen einer Gruppenordnung von 10^{40} unmöglich [1, S. 118]. Im Hinblick auf die von der Bundesnetzagentur empfohlene Schlüssellänge von 2^{2048} [5] oder höher ist von dieser Kategorie (1) der Algorithmen kaum eine Gefahr für die RSA Verschlüsselung ersichtlich.

3.2. Silver-Pohlig-Hellman Angriff

3.2.1. Algorithmus

Dieser Algorithmus wird nach der unter 2.3 beschriebenen Kategorisierung unter (2) zugeordnet und setzt damit einige charakterliche Eigenschaften des zu berechnenden diskreten Logarithmus voraus. Die wichtigste Eigenschaft ist dabei die Kenntnis der Primfaktorzerlegung der Gruppenordnung q, die ohnehin mit sehr hohem Aufwand verbunden ist, da noch kein effizienter Algorithmus für diese Thematik bekannt ist. Zusätzlich ist für die Laufzeit maßgebend, wie groß der größte Primfaktor der Zerlegung ist. Entsprechen alle Primfaktoren einem gleichmäßigen Muster (keine großen Primfaktoren), ist der Silver-Pohlig-Hellman Algorithmus besonders effizient [1, S. 118]. Pohlig und Hellman zeigten, dass

$$q - 1 = \prod_{i=1}^{k} p_i^{\alpha_i},$$

wobei p_i die Primfaktoren und die α_i natürliche Zahlen sind. Wenn $r_1, \ldots, r_k$ reelle Zahlen im Bereich $0 \leq r_i \leq 1$ darstellen, können Logarithmen über GF(q) in

$$O\left(\sum_{i=1}^{k} (\log q + p_i^{1-r_i}\,(1 + \log p_i^{r_i}))\right)$$

Operationen berechnet werden und benötigen

$$O\left(\log q \sum_{i=1}^{k} (1 + p_i^{r_i})\right)$$

Bits an Speicher, mit einer Vorberechnungszeit von

$$O\left(\sum_{i=1}^{k} p_i^{r_i} \log p_i^{r_i} + \log q)\right)$$

Operationen.

Es folgt eine Beschreibung des Algorithmus [1, S. 119] zur Berechnung des diskreten Logarithmus der Form $x = \log_a b \bmod q$:

(1) Faktorisiere $q - 1$ in dessen Primfaktoren:

$q - 1 = \prod_{i=1}^{k} p_i^{\alpha_i}$.

(2) Berechne die Tabelle $r_{p_{i,j}}$ für ein gegebenes Feld:

$r_{p_{i,j}} = a^{j(q-1)/p_i} \bmod q$, $0 \leq j \leq p_i$.

Dies muss einmalig für jedes gegebene Feld durchgeführt werden.

(3) Berechne den diskreten Logarithmus von b zur Basis a modulo q der Form $x = \log_a b \bmod q$:

(3-1) Finde die individuellen diskreten Logarithmen $x \bmod p_i^{\alpha_i}$, z. B. mit Hilfe des Baby-Step Giant-Step, indem folgendes angenommen wird:

$$x \bmod p_i^{\alpha_i} = x_0 + x_1 p_1 + \ldots + x_{\alpha_{i-1}} p_i^{\alpha_{i-1}},$$

wobei $0 \leq x_n < p_i - 1$.

(a) Um x_0 zu finden, berechnet man $b^{(q-1)/p_i}$, das zu $r_{p_i,j}$ für bestimmte j äquivalent ist, und setzt $x_0 = j$, sodass

$$b^{(q-1)/p_i} \bmod q = r_{p_i,j}.$$

Dies ist möglich, da

$$b^{(q-1)/p_i} \equiv a^{x(q-1)/p} \equiv a^{x_0(q-1)/p} \bmod q = r_{p_i,x_0}.$$

(b) Um x_1 zu finden, berechnet man $b_1 = ba^{-x_0}$. Falls ${b_1}^{(q-1)/p_i^2} \bmod q = r_{p_i,j}$, dann setze $x_1 = j$. Dies ist möglich, da

$$\begin{aligned} {b_1}^{(q-1)/p_i^2} &\equiv a^{(x - x_0)(q-1)/p_i^2} \equiv a^{(x_1 + x_2 p_i + \cdots)(q-1)/p_i} \\ &\equiv a^{x_1(q-1)/p} \equiv \bmod q = r_{p_i,x_1}. \end{aligned}$$

(c) Für die Bestimmung von x_2 muss man $b_2 = ba^{-x_0 - x_1 p_i}$ berücksichtigen und ${b_2}^{(q-1)/p_i^3} \bmod q$ berechnen.

Diese Vorgehensweise wird induktiv für alle zu bestimmenden x_0, x_1, ..., $x_{\alpha_i - 1}$ weitergeführt.

(3-2) Benutze das Chinesische Restsatz Theorem, um den einzigartigen Wert von x der Kongruenzen x mod $p_i^{\alpha_i}$ zu bestimmen.

3.2.2. Gefahrenpotential für RSA

Die Reduktion der Laufzeit einer auf Wurzelberechnung basierenden Methode (z. B. Baby-Step Giant-Step) $\approx O(\sqrt{q})$ auf $\approx O(\sqrt{p})$ [1, S. 118] steht hier eindeutig im Fokus des Algorithmus von Pohlig und Hellman. Die Tatsache, dass ein solcher Schritt durch die Bekanntheit der Primfaktorzerlegung der Gruppenordnung möglich ist, zeigt, dass in diesem Algorithmus viel Potenzial steckt, um RSA brechen zu können. Allerdings ist die Faktorisierung solcher Zahlen ein ebenso schwieriges Problem wie das DLP selbst. Daran lässt sich abermals erkennen, wie sehr das IFP und das DLP miteinander verbunden sind. Ließe sich das eine effizient lösen, wäre das andere damit auch gelöst wie im beschriebenen Algorithmus auch gezeigt. Solange dies auch weiterhin der Fall bleibt, ist an dieser Stelle auch keine direkte Gefahr für das RSA-Kryptosystem erkennbar.

3.3. Index Calculus Angriff

3.3.1. Algorithmus

Die letzte Kategorie von Algorithmen wird durch den Index Calculus (IC) repräsentiert. Es existieren mehrere Varianten dieser Methode, die in subexponentieller Zeit einen diskreten Logarithmus bestimmen können und somit zu den schnellsten der bekannten Algorithmen für das DLP zählen. Der IC [1, S. 122 f.] versucht eine ganze Zahl k zu finden, sodass

$$k \equiv \log_\beta \alpha \pmod{p} \text{ oder } \alpha \equiv \beta^k \pmod{p}.$$

(1) Vorberechnung

(1-1) Wähle eine Faktorbasis Γ, die aus den ersten m Primzahlen bestehen,

$$\Gamma = \{p_1, p_2, \ldots, p_m\},$$

mit $p_m \leq B$, der Schranke des Basis.

(1-2) Wähle eine zufällige Menge von Exponenten $e \leq p - 2$, berechne β^e mod p und faktorisiere die Primzahlpotenzen als Produkte.

(1-3) Sammle nur Relationen $\beta^e \bmod p$, die in Bezug auf B gleichmäßig sind:

$$\beta^e \bmod p = \prod_{i=1}^{m} p_i^{e_i}, e_i \geq 0.$$

Falls solche Relationen existieren, bestimme

$$e \equiv \sum_{j=1}^{m} e_j \log_\beta p_j \pmod{p-1}.$$

(1-4) Wiederhole den Schritt (1 – 3) solange, bis mindestens m solcher e's gefunden wurden, sodass m Relationen bestimmt werden konnten und berechne $\log_\beta p_j$, mit $j = 1, 2, \ldots, m$.

(2) Berechne $k \equiv \log_\beta \alpha \pmod{p}$:

(2-1) Bestimme für jedes e den Wert von $\log_\beta p_j$, mit $j = 1, 2, \ldots, m$.

(2-2) Wähle einen zufälligen Exponenten $r \leq p - 2$ und berechne $\alpha\beta^r \bmod p$.

(2-3) Faktorisiere $\alpha\beta^r \bmod p$ über Γ:

$$\alpha\beta^r \bmod p = \prod_{j=1}^{m} p_j^{r_j}, r_j \geq 0$$

Falls dies erfolglos war, gehe zu (2-2). Falls dies erfolgreich war, dann

$$\log_\beta \alpha \equiv -r + \sum_{j=1}^{m} r_j \log_\beta p_j.$$

3.3.2. Beispiel

Nun wird der unter 3.3.1 dargestellte Algorithmus in einem Beispiel angewendet, um die einzelnen Schritte im Speziellen zu erläutern [1, S. 123 f.]. Gesucht sei der diskrete Logarithmus:

$$x \equiv \log_{22} 4 \pmod{3361}, \text{ sodass } 4 \equiv 22^x \pmod{3361}.$$

(1) Vorberechnung

(1-1) Wähle eine Faktorbasis Γ, bestehend aus den ersten vier Primzahlen:

$$\Gamma = \{2, 3, 5, 7\},$$

mit $p_4 \leq 7$, der Schranke der Basis.

(1-2) Wähle eine zufällige Menge von Exponenten $e \leq 3359$, berechne 22^e mod 3361 und faktorisiere die Primzahlpotenzen als Produkte.

$22^{48} \equiv 2^5 \cdot 3^2 \pmod{3361}$

$22^{100} \equiv 2^6 \cdot 7 \pmod{3361}$

$22^{186} \equiv 2^9 \cdot 5 \pmod{3361}$

$22^{2986} \equiv 2^3 \cdot 3 \cdot 5^2 \pmod{3361}$

(1-3) Diese vier aufgestellten Relationen sind gleichmäßig in Bezug auf $B = 7$:

$48 \equiv 5 \log_{22} 2 + 2 \log_{22} 3 \pmod{3360}$

$100 \equiv 6 \log_{22} 2 + \log_{22} 7 \pmod{3360}$

$186 \equiv 9 \log_{22} 2 + \log_{22} 5 \pmod{3360}$

$2986 \equiv 3 \log_{22} 2 + \log_{22} 3 + 2 \log_{22} 5 \pmod{3360}$.

(2) Berechne $k \equiv \log_\beta \alpha \pmod{p}$

(2-1) Berechne

$\log_{22} 2 = 1100$

$\log_{22} 3 = 2314$

$\log_{22} 5 = 366$

$\log_{22} 7 = 220$

(2-2) Wähle einen zufälligen Exponenten $r = 754 \leq 3359$ und berechne $4 \cdot 22^{754}$ mod 3361.

(2-3) Faktorisiere $4 \cdot 22^{754}$ mod 3361 über Γ:

$4 \cdot 22^{754} \equiv 2 \cdot 3^2 \cdot 5 \cdot 7 \pmod{3361}$.

Daraus lässt sich x bestimmen:

$\log_{22} 4 \equiv -754 + \log_{22} 2 + 2 \log_{22} 3 + \log_{22} 5 + \log_{22} 7 \equiv 2200$,

was bedeutet, dass:

$22^{2200} \equiv 4 \pmod{3361}$.

3.3.3. Algorithmus nach Adleman

Eine weitere Variation des Index Calculus‘ entwickelte Adleman im Jahre 1979. Dieser versucht den diskreten Logarithmus $x = \log_a b$ mod q zu lösen, wobei a und b Erzeuger sind und q eine Primzahl ist. [1, S. 125 f.]

(1) Faktorisiere $q - 1$ in dessen Primzahlfaktoren:

$$q - 1 = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$$

(2) Führe für jedes $p_k^{\alpha_k}$ die folgenden Schritte durch, bis man m_l erhält:

- Finde r_i, s_i, sodass $a^{r_i} \bmod q$ und $ba^{s_i} \bmod q$ in Bezug auf die Schranke $2^{(\log q \log\log q)^{1/2}}$ gleichmäßig sind.

- Überprüfe, ob $ba^{r_i} \bmod q$ über dem endlichen Körper $\mathbb{Z}_{p_l^{\alpha_l}}$ abhängig ist von:

  $$\{a^{r_1} \bmod q, \dots, a^{r_i} \bmod q\}.$$

  Falls dies der Fall ist, berechne β_j's, sodass

  $$ba^{s_i} \bmod q \equiv \left(\sum_{j=1}^{i} \beta_j a^{r_i} \bmod q \right) \bmod p_l^{\alpha_l}$$

  und dann

  $$m_l = \left(\sum_{j=1}^{i} \beta_j r_j \right) \bmod p_l^{\alpha_l} - s_i.$$

(3) Berechne mit Hilfe des Chinesischen Restsatzes x:

$$x \equiv m_l \pmod{p_l^{\alpha_l}},\ l = 1, 2, \dots, k.$$

Diese Methode kann auch auf einfache Weise generalisiert werden, wodurch q nicht zwingend eine Primzahl ist oder a oder b keine Erzeuger sind.

3.3.4. Number Field Sieve nach Gordon

Der Index Calculus galt lange Zeit als der schnellste Algorithmus, um diskrete Logarithmen zu lösen, bis Gordon 1993 eine Methode basierend auf dem NFS entwickelte [1, S. 126]. Dieser lässt sich wie folgt kurz skizzieren:

Dieser Algorithmus berechnet den diskreten Logarithmus x, sodass $a^x \equiv b \pmod p$, wobei a und b Erzeuger sind und p eine Primzahl:

(1) Finde die diskreten Logarithmen einer Faktorbasis mit kleinen Primzahlen. Dies muss nur einmalig für ein gegebenes p erfolgen.

(2) Finde die Logarithmen für jedes $b \in F_p$, indem die Logarithmen zu „mittelgroßen“ Primzahlen gefunden werden.

(3) Kombiniere alle individuellen Logarithmen (mit Hilfe des Chinesischen Restsatzes), um die Logarithmus von b zu erhalten.

3.3.5. Gefahrenpotential für RSA

Da dieser Algorithmus und dessen Varianten die Grundlage für anderen Methoden zur Berechnung des diskreten Logarithmus‘ liefern, nimmt der Index Calculus eine besondere Rolle ein. Mit einer heuristischen Laufzeit von:

$$O\left(\exp\left(c(\log p)^{\frac{1}{3}}\ (\log\log p)^{\frac{2}{3}}\right)\right)$$

zählt insbesondere das NFS nach Gordon zu den schnellsten Methoden, sowohl für das IFP als auch das DLP, da beide Probleme eng miteinander verknüpft sind. Somit ist der IC der zurzeit effizienteste Algorithmus zur Lösung von diskreten Logarithmen, gleichzeitig aber nicht effizient im Sinne der Komplexitätstheorie. Damit lässt sich auch im Hinblick auf die Sicherheit von RSA keine ernstzunehmende Gefahr ausmachen. Durch eine korrekte Verwendung der Schlüsselgrüße und ihrer Eigenschaften ist und bleibt die Public-Key Verschlüsselung eine der sichersten Verfahren für den Schutz von Daten und deren Austausch.

4. Fazit und Ausblick

In dieser Arbeit wurden unterschiedliche Algorithmen aufgezeigt, mit denen man direkt oder indirekt das DLP im begrenzten Rahmen lösen und damit sowohl die Public-Key Kryptographie im Allgemeinen als auch das RSA-Kryptosystem im Speziellen versucht zu brechen. Darunter befanden sich der Shanks‘ Baby-Step Giant-Step, der Silver-Pohlig-Hellman Algorithmus und verschiedene Varianten des Index Calculus‘, zum Beispiel nach Adleman.

Es wurde deutlich, dass der diskrete Logarithmus aufgrund der Eigenschaft der Einwegfunktion dafür geeignet ist, als Grundlage für die Sicherheit in kryptographischen Algorithmen eingesetzt zu werden. Dem diskreten Logarithmus kommt somit in der Kryptographie eine große Bedeutung zu. Schließlich beruht damit auch die Sicherheit der zu schützenden Informationen auf dem DLP. Bis heute ist kein effizienter Algorithmus bekannt, der das DLP lösen kann. Dennoch ist es nicht undenkbar, dass in der Zukunft ein Algorithmus gefunden wird, der das DLP (und das IFP im gleichen Zuge oder umgekehrt) in effizienter Zeit löst. Die dargestellten Methoden liefern wertvolle Grundlagen, auf denen eine potentielle Lösung beruhen kann.

5. Glossar

Notation	Erklärung
$\mathbb{Z}^+$	Menge der positiven ganzen Zahlen: $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$
$\mathbb{Z}_{>1}$	Menge der positiven ganzen Zahlen größer 1
$\mathbb{Z}^*_p$	Multiplikative Gruppe
F_p	Endliche Körper mit p Elementen, wobei p eine Primzahl ist
$[x]$	Größte Ganzzahl kleiner oder gleich x
$\phi(N)$	Eulersche Phifunktion
M	Klartext
C	Verschlüsselter Text
e	RSA Kodierungsexponent
d	RSA Dekodierungsexponent

6. Literaturverzeichnis

[1] Song Y. Yang, Springer US „*Cryptanalytic Attacks on RSA“*, S. 111 – 133, (2008)

[2] J. M. Pollard, American Mathematical Society „*Monte Carlo Methods for Index Computation (mod p)“*, S. 918 – 924, (1978)

[3] J. M. Pollard, Journal of Cryptology „*Kangaroos, Monopoly and Discrete Logarithms“*, S. 437 – 447, (2000)

[4] Victor Shoup, Cambridge University Press *„A Computational Introduction to Number Theory and Algebra“*, S. 327 – 334, (2008)

[5] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: *„Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen“*, S. 7, (21.01.2014)

[6] S. Pohlig and M. Hellman, IEEE Trans. Information Theory 24 *„An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance“*, S. 106 – 110, (1978)

[7] L. Adleman, In 20th Annual Symposium on Foundations of Computer Science *„A subexponential algorithm for the discrete logarithm problem with applications to cryptography“*, (1979)

[8] Daniel M. Gordon, Department of Computer Science *„Discrete Logarithms in GF(p) using the Number Field Sieve“*, (1992)