Seminar IT-Sicherheit

Cryptanalytic Attacks on RSA

1.1 Introduction & 1.2 Computability, Complexity and Intractability

von

Toni Rein

(winf100726@fh-wedel.de)

im

Sommersemester 2016

Dozent:

Prof. Dr. Gerd Beuster

(gb@fh-wedel.de)

Eingereicht am 22.05.2016

Inhalt

1. Einleitung

Immer wieder stellt sich die Frage nach der Laufzeit eines Algorithmus und wie schnell oder effizient dieser ist. Um die Laufzeit zu bestimmen, verwenden Mathematiker und Informatiker keine zeitliche Angabe in Millisekunden, Sekunden oder Minuten, sondern elementare Rechenschritte. Die Laufzeit hängt von der Größe der Input-Daten ab und wie viele elementare Rechenschritte nötig sind. So lässt sich beispielsweise der kürzeste Weg von der FH-Wedel zur akquinet AG in Hamburg mit weniger Daten berechnen, als der Weg von Rom nach Berlin. Ähnliches gilt für das Rechnen mit Zahlen. Während das Multiplizieren mit einstelligen Zahlen in einem elementaren Rechenschritt funktioniert, benötigt der Rechner deutlich mehr für die Multiplikation zweier hundertstelliger Zahlen.

Schnell wird klar, dass es Probleme gibt, welche nicht effizient oder schnell berechnet werden können, wie zum Beispiel das „Traveling Salesman Problem“. Diese Probleme befinden sich in der Klasse „*NP*“ und gelten als schwierig bis unmöglich zu lösen.

Diese Art von Problemen kann aber von Nutzen sein. Sämtliche Public-Key-Verschlüsselungsverfahren beruhen darauf, dass die privaten Schlüssel nicht effizient berechnet werden können, um die gesendeten Daten zu entschlüsseln. Diese Seminararbeit beschäftigt sich mit Problemarten, Komplexitätsklassen und der Verbindung zum RSA-Problem.

2. Zahlentheoretische Probleme

Um ein Verständnis für Komplexitätsklassen und deren Verbindung zu RSA zu bekommen, werden zunächst die relevanten Problemarten erläutert. Insbesondere schwer lösbare, zahlentheoretische Probleme sind stark mit „Public-Key-Kryptografie“ verbunden. Auch das RSA Verschlüsselungsverfahren basiert darauf. Die Sicherheit sämtlicher Public-Key-Kryptografien ist in irgendeiner Form von schwer lösbaren, zahlentheoretischen Problemen abhängig. Nur aus diesem Grund lassen sich Daten somit sicher mit dem RSA-Verfahren verschlüsseln. [3]

2.1. Problemarten

Probleme können in folgende Kategorien aufgeteilt werden:

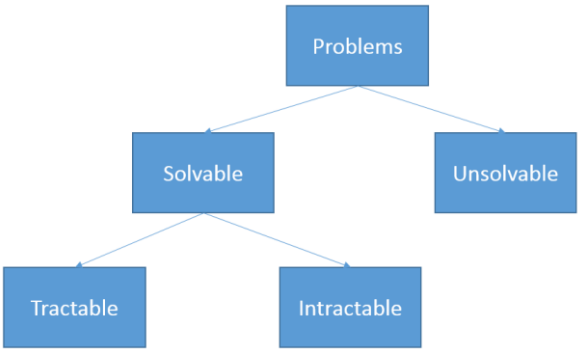

Abbildung 1: Aufteilung von Problemarten

Solvable (lösbare Probleme): wenn ein Problem durch eine Turingmaschine gelöst werden kann, nennt man diese lösbar, andernfalls unlösbar (unsolvable).

Tractable (einfach lösbar): wenn ein Problem in polynomieller Zeit durch eine deterministische Turingmaschine gelöst werden kann, nennt man diese einfach lösbar (tractable), andernfalls schwer lösbar (intractable).

Probleme, die in polynomieller Zeit auf Turingmaschine gelöst werden, sind identisch mit denen, die auf einem typischen Computer gelöst werden können.

Da es keine klare Abgrenzung zwischen tractable und intractable Problemen gibt, werden diese durch die Komplexitätsklassen *P* und *NP* unterschieden. Für

die RSA Public-Key-Kryptografie sind beide Problemarten relevant. Beispielsweise muss die Entschlüsselung von Daten ohne private Key sehr aufwändig (intractable) sein, aber die Verschlüsselung mit dem public key einfach (tractable) sein. [3]

2.2. Algorithmen für tractable Probleme

Folgende effiziente Algorithmen spielen eine wichtige Rolle in vielen Public-Key-Kryptografien. Sie gehören zu den tractable Problemen und werden, bis auf den euklidischen Algorithmus, nur genannt:

- Euklidischer Algorithmus
- Binäre Modulo-Exponentiation
- Chinesischer Restsatz
- Effiziente Algorithmen für Primzahltests

Der euklidische Algorithmus wird später im Skript genauer erläutert, siehe Kapitel 4.2. [3]

2.3. Beispiele für intractable Probleme

Folgende Probleme dienen als Beispiele, warum Public-Key-Kryptografien als sicher gelten, da die Lösung dieser Probleme sehr schwer ist:

- Primfaktorzerlegung (für RSA)
- Diskrete Logarithmus Probleme (für Diffie-Hellman-Merkle Schlüsselaustausch)
- Quadratisches Restproblem (für Goldwasser-Micali Verschlüsselung)
- Modular Root finding problem (für Rabin Kryptografie) [3]

3. Berechenbarkeit und Komplexitätsklassen

3.1. Turingmaschinen

Turingmaschinen, welche von Alan Turing 1936 vorgestellt wurden, bieten die einfachste Möglichkeit moderne Computer darzustellen. Zunächst erfolgt eine formale Definition einer deterministischen Turingmaschine (DTM):

$M = (Q, \sum, \Gamma, \delta, q_0, \square, F)$

Γ = eine endliche Menge, das Bandalphabet. $\Gamma \neq 0$

δ = Übergangsfunktion (oder Turingmaschinen-Programm):

- wenn M eine DTM ist, dann $\delta: Q \times \Gamma k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$
- wenn M eine nichtdeterministische Turingmaschine (NDTM) ist, dann $\delta: Q \times \Gamma k \rightarrow 2^{Q \times \Gamma^k \times \{L,R\}^k}$

$\sum$ ist das Eingabealphabet $\sum \subseteq \Gamma - \{\square\}$

Q ist eine endliche Zustandsmenge

q_0 ist der Startzustand $q_0 \in Q$

$\square$ ist das Leerzeichen $\square \in \Gamma \setminus \sum$

F ist die Menge der Endzustände $F \subseteq Q$

Definition für berechenbare Funktionen: Sofern eine Funktion auf einer Turingmaschine berechnet werden kann, nennt man diese eine berechenbare Funktion, andernfalls unberechenbar. Eine Sprache wird akzeptiert, wenn diese auf einer Turingmaschine akzeptiert wird, andernfalls inakzeptabel.

Die NDTM ist eine Erweiterung der DTM und hat zu jedem Zeitpunkt mehrere Möglichkeiten eine Berechnung fortzusetzen, wodurch der Rechenweg nicht eindeutig bestimmt werden kann. Die NDTM ist nur ein theoretisches Modell und ist nicht gleichwertig zu heutigen Computern. Es ist notwendig um weitere Komplexitätsklassen zu unterscheiden. [2] [3]

3.2. Church-Turing These

„Any effectively computable function can be computed by a Turing machine."

- **Alonzo Church / Alan Turing: An unsolvable Problem of Elementary Number Theory 1936**

Jede effektiv berechnete Funktion kann demnach durch eine Turingmaschine berechnet werden. Mit Hilfe dieser These ist es möglich zu bestimmen welche Funktion berechenbar ist, oder allgemein gehalten: was ein Computer kann, oder nicht kann.

Wichtig: Die Church-Turing These ist kein mathematisches Theorem und kann somit auch nicht bewiesen werden. Um die These zu beweisen, müsste zunächst spezifiziert werden, was überhaupt effektiv berechenbar ist. Dies ist unmöglich. Allerdings wird die These durch vielfältige Nachweise unterstützt und bisher konnte die Church-Turing These nicht widerlegt werden.

Aus Sicht der Informatik und besonders der Kryptografie, ist es nicht nur wichtig zu unterscheiden, ob Funktionen berechnet werden können oder nicht. Viel wichtiger ist die Frage, wie schnell und effizient eine Funktion berechnet werden kann. Besonders in der Kryptografie ist diese Frage von Bedeutung. [3]

3.3. Komplexitätsklassen

Anhand der Funktionsweise von Turingmaschinen, können einige Komplexitätsklassen definiert werden. Zunächst eine Definition für probabilistische oder zufällige Turingmaschinen:

Probabilistische Turingmaschine (PTM):

- Gehört zu den NDTM und hat eindeutige Zustände, genannt „coin-tossing" Zustände
- Für jeden dieser „coin-tossing" Zustände gibt das Steuerwerk (finite control) genau 2 mögliche Zustände

- Berechnung ist deterministisch, außer während den „coin-tossing“ Zuständen, da dort eine 50/50 Entscheidung getroffen wird, um zum nächsten Zustand zu gelangen
- Kann auch als zufällige Turingmaschine angesehen werden [3]

3.3.1. Komplexitätsklassen P und NP

Definition *P*: Klasse von Problemen, welche in polynomieller Zeit durch eine DTM gelöst werden können. Außerdem gibt es ein Polynom n^k (k ist eine Konstante), welches die Laufzeit abhängig von der Eingabe *n* beschreiben kann. In dieser Klasse sind die Probleme tractable und durch einen Computer einfach zu lösen. Beispiele für Probleme in *P* sind: Addieren zweier Integer kann in polynomieller Zeit gelöst werden, egal wie groß die Stellenanzahl der Integer ist, oder das Sortierproblem in dem beispielsweise ein Array sortiert wird.

Definition *NP:* Klasse von Problemen, bei denen die Lösung in polynomieller Zeit verifiziert werden kann, aber die Berechnung häufig exponentielle Zeit benötigt. In dieser Klasse sind die Probleme intractable und durch einen Computer schwer zu lösen. Als Beispiel dient das „Traveling Salesman Problem“. Es wird der kürzeste Weg gesucht bei dem jeder Knoten genau einmal besucht werden muss und der Anfangsknoten gleich dem Endknoten ist.

In Bezug auf formale Sprachen gibt es folgenden Unterschied zwischen *P* und *NP*. Wenn die Zugehörigkeit einer formalen Sprache in polynomieller Zeit bestimmt werden kann, gehört sie zu der Klasse von *P*. Wenn die Zugehörigkeit hingegen nur nachgewiesen werden kann, gehört die Sprache zur Klasse von *NP*. Mit nachweisen ist gemeint, dass die NDTM versucht eine Lösung zu „erraten“ und jeder Lösungsversuch in polynomieller Zeit versucht verifiziert zu werden. Somit gibt es zu einer Eingabe nicht nur eine Berechnung (deterministisch), sondern mehrere mögliche Berechnungen (nichtdeterministisch). Als eines der größten, ungelösten Probleme gilt die Frage ob *P* = *NP* ist. Diese Frage wurde in die Liste der Millennium-Probleme aufgenommen.

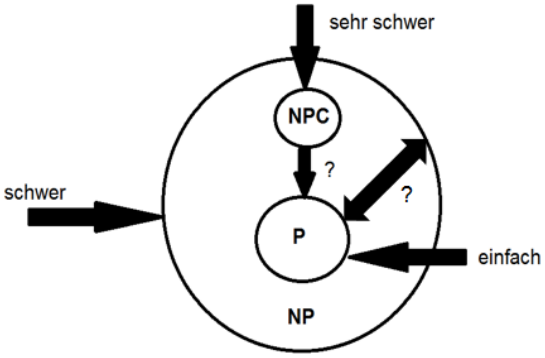

Abbildung 2: P-NP Problem

Definition *EXP* : Klasse von Problemen, welche durch eine DTM gelöst werden kann, mit einer zeitlichen Schranke von 2^{n^i}. Die schwersten Probleme dieser Klasse können also nur in exponentieller Zeit gelöst werden. Dadurch, dass jedes Problem, welches in polynomieller Zeit gelöst werden kann, auch unterhalb der zeitlichen Schranke von EXP gelöst werden kann, gilt: $P \subset EXP$.

Eine Funktion *f* ist in polynomieller Zeit berechenbar, wenn für irgendeine Eingabe *w* gilt*: f(w)* stoppt auf einer Turingmaschine in polynomieller Zeit. Eine Sprache *A* ist in polynomieller Zeit zu einer Sprache *B* reduzierbar (geschrieben: $A \leq_p B$), wenn gilt: es existiert eine in polynomieller Zeit berechenbare Funktion, welche für jeden Input *w*: $w \in A \Leftrightarrow f(w) \in B$ (dies nennt man polynomielle Zeitreduktion von *A* zu *B*).

***NP-schwer*:** Ein Problem *D* nennt man *NP-schwer*, wenn es die Bedingung erfüllt: $\forall A \in NP,\ A \leq_p D$. Das bedeutet, dass jedes Problem aus *NP* sich in Polynomialzeit auf *D* reduzieren lässt (soll heißen in *P*). *D* kann in *NP* liegen, muss es aber nicht. *NP-schwer* heißt, dass es mindestens so schwer ist wie *NP*, aber es kann durchaus noch schwieriger sein.

***NP-vollständig*:** Wenn ein Problem *D NP-schwer* ist und zusätzlich noch in *NP* liegt, dann nennt man dieses *NP-vollständig*. Da die Reduktion die eben definierte Funktion *f* ist, kann getestet werden, ob für Input *w*: $w \Leftrightarrow f(w)$ gilt. Mittlerweile sind viele Probleme bekannt, die NP-vollständig sind. Es genügt nur eines davon effizient, also in polynomieller Zeit, zu lösen. In diesem Fall wäre bewiesen, dass *P = NP* ist, da jedes *NP-vollständige* Problem sich in eines in *P* reduzieren lässt. [3] [4] [5]

3.3.2. Weitere häufig verwendete Komplexitätsklassen

In ähnlicher Weise können die Klassen der Probleme *P-Space, P-Space-vollständig* und *P-Space-Schwe*r beschrieben werden. Folgende Abkürzungen werden verwendet:

- *NPC*: NP-vollständig (NP-Complete)
- *PSC*: P-Space-vollständig (P-Space-Complete)
- *NPH*: NP-schwer (NP Hard)
- *PSH*: P-Space-schwer (P-Space Hard)

Die Beziehungen zu den Klassen *P, NP, NPC, PSC, NPH, PSH* und *EXP* werden in Abbildung 3 beschrieben.

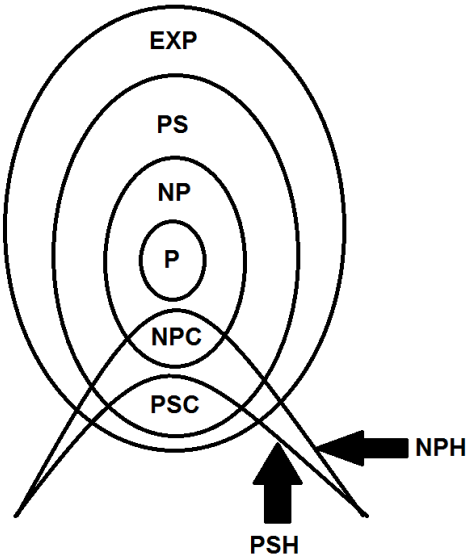

Abbildung 3: Beziehung zwischen den Komplexitätsklassen P, NP und NPC, etc.

RP: Klasse von Problemen, welche in einer erwarteten polynomiellen Zeit berechnet werden können mit einem einseitigen Fehler, durch eine probabilistische Turingmaschine. Einseitiger Fehler bedeutet, dass die Turingmaschine mit „Ja“ antwortet und „Ja“ eine Fehlerwahrscheinlichkeit von unter 50% hat. Die Antwort „Nein“ hingegen ist immer richtig.

ZPP: Klasse von Problemen, welche in einer erwarteten polynomiellen Zeit berechnet werden, ohne Fehler auf einer probabilistischen Turingmaschine. *ZPP* ist definiert als: *ZPP = RP ∩ co-RP. co-RP* ist das Komplement von *RP*. Die Antworten „Ja“ und „Nein“ werden ohne Fehler ausgegeben, allerdings kann die Turingmaschine auch ein “?“ ausgeben. Das heißt, die Turingmaschine kennt

die Antwort nicht. Es ist garantiert, dass die Antwort „?“ nicht häufiger als 50% der Fälle auftritt.

BPP: Klasse von Problemen, welche in erwarteter polynomieller Zeit berechnet werden, mit einem zweiseitigen Fehler auf einer PTM. Die Antwort hat immer eine Wahrscheinlichkeit von $½ + \sigma$ richtig zu sein, mit $\sigma > 0$. Das *B* in *BPP* steht für: „bounded away the error probability from ½”. Die Fehlerwahrscheinlichkeit liegt zum Beispiel bei 1/3.

Die Klassen *P-Space* und *NP-Space* lassen sich wie *P* und *NP* definieren. Obwohl es nicht bekannt ist ob *P=NP* ist, wurde bewiesen das *P-Space = NP-Space*, durch den Satz von Savitch. Folgende Beziehungen werden angenommen:

$P \subseteq ZPP \subseteq RP \subseteq (BPP\ NP) \subseteq P\text{-}Space \subseteq EXP$

Allerdings ist es nicht sicher, ob diese Annahme richtig ist. Nur $P \subset EXP$ ist bekannt. Die Beziehung zwischen *BPP* und *NP* ist unbekannt. Es wird angenommen, dass $NP \not\subseteq BPP$ ist. [3] [4]

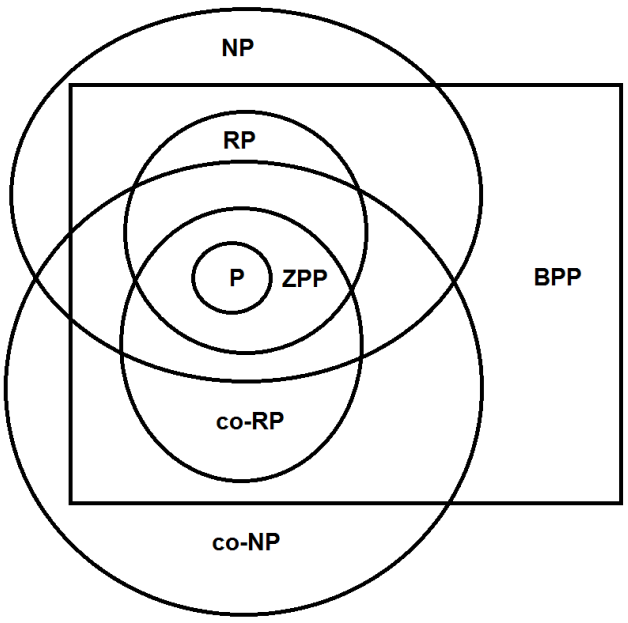

Abbildung 4: Beziehung zwischen einigen, häufigen Komplexitätsklassen

3.4. Cook-Karp These

„Any computationally tractable problem can be computed by a Turing machine in deterministic polynomial-time. “

- **Stephen Cook / Richard Karp: The complexity of Theorem-Proving Procedures**

Es wird angenommen, dass Probleme die in *P* liegen einfach zu lösen sind und Probleme, die nicht in *P* liegen, schwer lösbar sind. Dies führte zu der Cook-Karp These, welche nach Stephen Cook und Richard Karp benannt wurde.

Auch hier gibt es keine klare Trennung zwischen den Arten der beiden Probleme. Dies entspricht wieder dem *P-NP* Problem. Auch diese These ist nur eine mathematische Theorie und nicht bewiesen, allerdings gibt es Nachweise, welche die These unterstützen. [3]

4. Verbindung zum RSA-Problem

In der RSA Public-Key-Verschlüsselung wird eine Nachricht *M* durch folgende Formel verschlüsselt: $C = M^e \pmod n$. Um alle benötigten Werte für diese Formel zu erhalten, sind folgende Schritte notwendig:

1. Wähle zwei unabhängige Primzahlen (*p,q*) aus und berechne $n = p * q$.
2. Berechne die Eulersche Phi-Funktion $\varphi(n) = (p-1)*(q-1)$. Diese wird für die Berechnung des privaten und öffentlichen Schlüssels benötigt.
3. Der Verschlüsselungsexponent *e* ist eine teilerfremde Zahl von $\varphi(n)$, außerdem muss gelten: $1 < e < \varphi(n)$.
4. Um den Entschlüsselungsexponent *d* zu errechnen, muss das multiplikative Inverse zu *e* bezüglich des Moduls $\varphi(n)$ berechnet werden, so dass die folgende Kongruenz gilt: $e * d \equiv 1 \pmod{\varphi(n)}$. Das bedeutet, dass sich *e* und *d* gegenseitig aufheben. [1]

4.1. Erweiterter euklidischer Algorithmus

Der erweiterte euklidische Algorithmus ist ein Verfahren, um den größten gemeinsamen Teiler (ggT) zweier positiver Ganzzahlen zu berechnen. Zusätzlich werden zwei ganze Zahlen *s* und *t* berechnet, damit folgende Gleichung erfüllt werden kann: $ggT(a, b) = s * a + t * b$. Auf diese Weise lassen sich auch Inverse in der Modulorechnung bestimmen, welche für das RSA-Verfahren benötigt werden.

Um das multiplikative Inverse zu lösen, welches in Schritt 4 angesprochen wurde, verwendet man den erweiterten euklidischen Algorithmus. Dazu wird zunächst folgende Formel aufgestellt:

$e * d + k * \varphi(n) = 1 = ggT(e, \varphi(n))$.

Nun können die Faktoren *d* und *k* berechnet werden. Wichtig für das RSA-Verfahren ist hierbei nur *d*. Den Faktor *k* benötigt man nicht. Anhand eines Beispiels für die Primzahlen $p = 13$ und $q = 11$ lässt sich auch die Berechnung des erweiterten euklidischen Algorithmus zeigen.

1. $n = p * q = 11 * 13 = 143$
2. $\varphi(n) = (p - 1) * (q - 1) = 10 * 12 = 120$
3. $e = 23$ *und es gilt* $1 < 23 < 120$
4. $e * d + k * \varphi(n) = 1 = 23 * d + 120 * k$

Nun wird der Algorithmus verwendet, um *d* und *k* zu bestimmen[1]:

A	B	Q	R	S	T	U	V	Berechnung
120	23			1	0	0	1	Startwerte
120	23	5	5	0	1	1	-5	$Q = A / B = 120 / 23 = 5$ $R = A \% B = 120 \% 23 = 5$ $S = U_{alt} = 0$ $T = V_{alt} = 1$ $U = S_{alt} - (Q \cdot U_{alt}) = 1 - (5 \cdot 0) = 1$ $V = T_{alt} - (Q \cdot V_{alt}) = 0 - (5 \cdot 1) = -5$
23	5	4	3	1	-5	-4	21	$Q = A / B = 23 / 5 = 4$ $R = A \% B = 23 \% 5 = 3$ $S = U_{alt} = 1$ $T = V_{alt} = -5$ $U = S_{alt} - (Q \cdot U_{alt}) = 0 - (4 \cdot 1) = -4$ $V = T_{alt} - (Q \cdot V_{alt}) = 1 - (4 \cdot -5) = 21$
5	3	1	2	-4	21	5	-26	$Q = A / B = 5 / 3 = 1$ $R = A \% B = 5 \% 3 = 2$ $S = U_{alt} = -4$ $T = V_{alt} = 21$ $U = S_{alt} - (Q \cdot U_{alt}) = 1 - (1 \cdot -4) = 5$ $V = T_{alt} - (Q \cdot V_{alt}) = -5 - (1 \cdot 21) = -26$
3	2	1	1	5	-26	-9	47	$Q = A / B = 3 / 2 = 1$ $R = A \% B = 3 \% 2 = 1$ $S = U_{alt} = 5$ $T = V_{alt} = -26$ $U = S_{alt} - (Q \cdot U_{alt}) = -4 - (1 \cdot 5) = -9$ $V = T_{alt} - (Q \cdot V_{alt}) = 21 - (1 \cdot -26) = 47$
	1			-9	47			Endergebnis

Abbildung 5: Berechnung des euklidischen Algorithmus

Mit dieser Berechnung ist $d = 47$ ermittelt. Um jetzt eine Nachricht zu verschlüsseln, wird durch den öffentlichen Schlüssel *e* und *n* beispielsweise die Nachricht 7 verschlüsselt. Es ergibt sich $C \equiv 7^{23} \pmod{143}$. Die verschlüsselte Nachricht ergibt damit $C = 2$. Mit Hilfe des privaten Schlüssels (*n, d*) kann die Nachricht einfach entschlüsselt werden: $M = C^d \pmod{n}$. $M \equiv 2^{47} \pmod{143}$ und *M* ergibt wieder 7. Die Nachricht wurde somit entschlüsselt. [1]

[1] Diese Berechnung wurde mit Hilfe des Rechners auf folgender Seite durchgeführt: http://www.johannes-bauer.com/compsci/eea/

4.2. RSA-Problem

Ein Dritter kann durch Abhören die verschlüsselte Nachricht *C* erlangen und kann ebenfalls den öffentlichen Schlüssel besitzen. Trotzdem sollte es dem Dritten nicht möglich sein die Nachricht zu entschlüsseln, ohne den private key zu besitzen. Die Aufgabe des Dritten lässt sich als RSA-Problem beschreiben:

Gegeben ist ein öffentlicher Schlüssel (*n, e*) und eine verschlüsselte Nachricht $C = M^e \ (mod\ n)$, berechne *M*.

Die Schwierigkeit, die sich einem Angreifer stellt, ist es die beiden Primzahlen zu ermitteln, welche genutzt wurden um *n* zu errechnen. Wie in den obigen Schritten zu sehen ist, wird für die Bestimmung des öffentlichen Schlüssels *e* und privaten Schlüssels *d*, die Eulersche Phi-Funktion verwendet, welche mit den Primzahlen *p* und *q* berechnet wird. Herausgegeben wird aber nur *n*, das Produkt von zwei Primzahlen. Somit muss eine Primfaktorzerlegung vorgenommen werden, um beide Primzahlen zu ermitteln. Die Primfaktorzerlegung zählt zu den intractable Problemen, gilt somit als schwer zu lösen und liegt in der Klasse *NP*.

Das gesamte RSA-Verfahren stützt sich also genau auf die Sicherheit, dass die Primfaktorzerlegung bis heute noch nicht effizient gelöst werden kann. Da die Dauer von dem Input, also den beiden Primzahlen abhängt, müssen diese ausreichend groß und unabhängig voneinander gewählt werden. Die Bundesnetzagentur empfiehlt Primzahlen so zu wählen, dass ein Schlüssel mit einer Länge von mindestens 2048 [2]Bit entsteht. Wie lange dieser Schlüssel sicher bleibt, oder ob das RSA-Verfahren als solches überhaupt sicher bleibt, hängt also davon ab, ob in Zukunft bessere Rechner mit mehr Rechenleistung erfunden werden, oder ob es einen Algorithmus geben wird, der die Primfaktorzerlegung in polynomieller Zeit ermöglicht und somit womöglich auch die Frage geklärt wird, ob *P* = *NP* ist. [1]

[2] Quelle: https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/QES/Veroeffentlichungen/Algorithmen/2014Algorithmenkatalog.pdf?__blob=publicationFile

5. Fazit und Ausblick

Probleme, die bis heute noch nicht effizient gelöst werden können, müssen nicht unbedingt etwas Negatives sein. Ganz im Gegenteil, aufgrund von intractable Problemen ist es möglich, Nachrichten und Signaturen zu verschlüsseln, um Sicherheit zum Beispiel beim Austausch von Nachrichten zu gewährleisten. Auch das RSA-Verfahren basiert auf dieser Sicherheit und setzt voraus, dass es keine effiziente Lösung für Probleme gibt, die in der Komplexitätsklasse NP-Vollständig liegen. Falls es jemals einen Rechner geben wird, der genug Rechenleistung besitzt, um 2048 Bit lange Schlüssel zu entschlüsseln, können größere Primzahlen gewählt werden, um so wieder Sicherheit zu gewährleisten. Zwar ist es bereits heutzutage ein Problem, das es sich mit großen Schlüsseln langsam rechnet, allerdings ist die Erzeugung der Schlüssel deutlich schneller und effizienter, als das Entschlüsseln ohne private Key. Falls aber jemals die Frage gelöst wird, ob $P = NP$ ist und somit ein effizienter Algorithmus gefunden wird, der auf weitere Probleme in NP angewandt werden kann, müssen andere Verfahren gefunden werden, um eine Verschlüsselung von Nachrichten zu ermöglichen.

6. Abbildungsverzeichnis

7. Literaturverzeichnis

[1] Ronald L. Rivest, Burt Kaliski (10.12.2003), „RSA Problem“ https://people.csail.mit.edu/rivest/RivestKaliski-RSAProblem.pdf

[2] A. M. Turing (12.11.1936) “ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM” https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

Aufruf: 04.05.2016

[3] Yan, Song Y. (2008) “Cryptanalytic Attacks on RSA” (S. 1-15), Springer US

[4] Martin Grötschel (13.09.2005) “Das Problem mit der Komplexität: P = NP?“ https://www.zib.de/groetschel/pubnew/paper/groetschel2007b_pp.pdf

[5] Larry Hardesty (29.10.2009) http://news.mit.edu/2009/explainer-pnp

Aufruf: 04.05.2016