Heinrich-Heine-Gymnasium

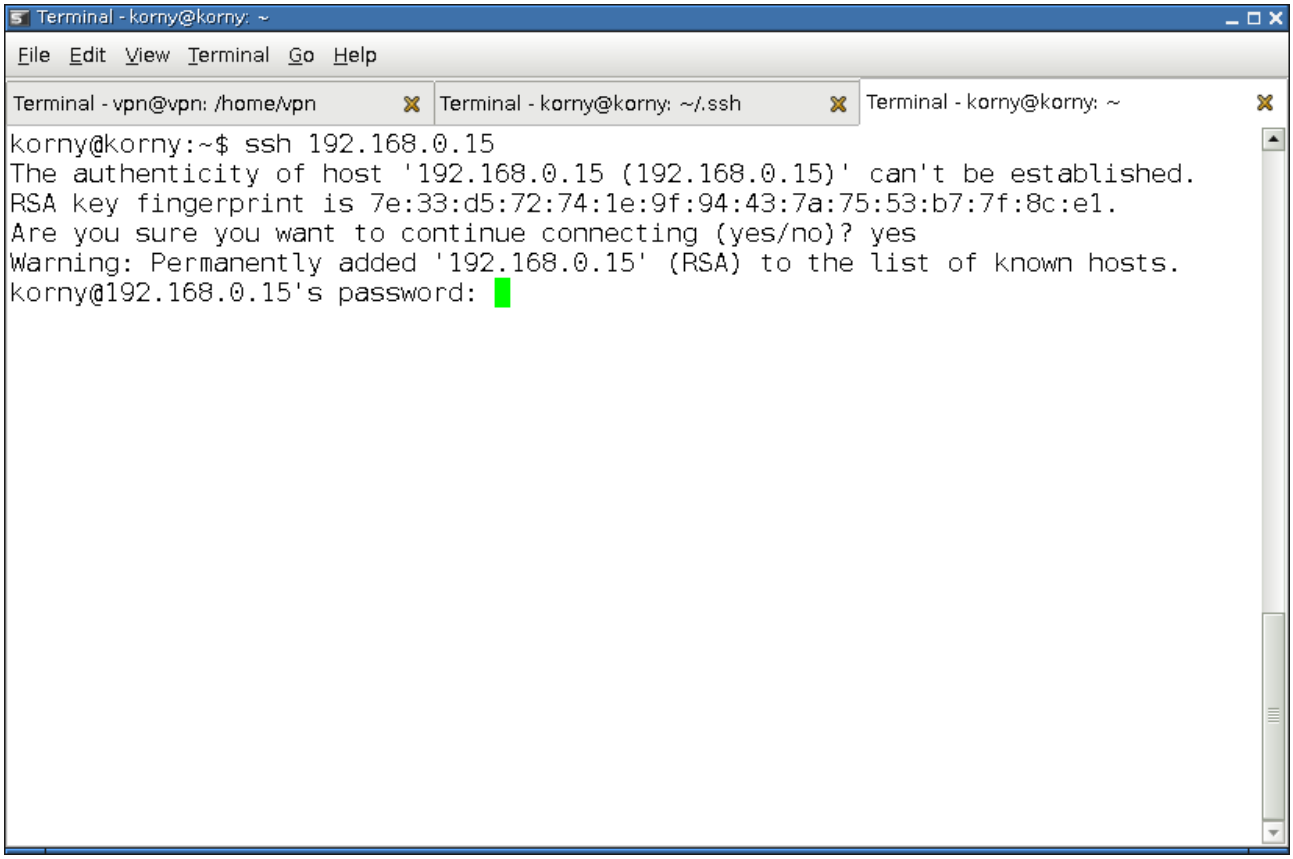
Facharbeit

aus dem Fach

Mathematik

Thema:

Mathematik der Kryptographie



Verfasser: Cornelius Diekmann

Kursleiter: Frau Nikles

Erzielte Note: ..... in Worten: .....

Erzielte Punkte: ..... in Worten: .....

Abgegeben beim Kollegstufenbetreuer: .....

.....

Unterschrift der Kursleiterin

Inhaltsverzeichnis

1 **Zu dieser Arbeit**

Diese Arbeit beschäftigt sich mit den mathematischen Aspekten der Kryptologie im sogenannten digitalen Zeitalter. Zunächst möchte ich erklären, warum Kryptologie heutzutage eine so wichtige Rolle einnimmt oder einnehmen sollte. Deswegen wird diese Arbeit sich häufig auf eine der meist genutzten Errungenschaft des digitalen Zeitalters beziehen: Der Vernetzung unabhängiger Rechenmaschinen, dem Internet. Danach sollen einige wichtige Grundbegriffe der Kryptologie erläutert werden und darauf folgt die Anwendung im RSA-Algorithmus.

Die Arbeit besteht aus zwei Teilen: Der Ausarbeitung des Themas „Mathematik der Kryptologie“ und einem extra beigelegten Anhang. Im Anhang finden sich erweiterte Erklärungen und genauere Details, die aus der eigentlichen Arbeit entnommen wurden, um den Lesefluss nicht zu stören.

Des Weiteren hab ich in dieser Arbeit zur besseren Veranschaulichung zwei frei erfundene Personen verwendet. Sie heißen Tim und Tux[1]. Wie es in gängigen Programmiersprachen üblich ist, leitet ein Rautesymbol (#) ein zusätzlich erläuterndes Kommentar innerhalb einer Rechnung bis zum Zeilenende ein.

2 **Einführung in die Mathematik der Verschlüsselung**

Prinzipiell gibt es zwei Arten der Verschlüsselung: Symmetrische Verschlüsselungssysteme und asymmetrische Verschlüsselungssysteme. Erstere werde ich grundlegend anhand eines geschichtlichen Beispiels im ersten Teil dieser Facharbeit vorstellen, letztere bilden den zweiten Abschnitt dieser Arbeit, denn Vorwissen aus dem ersten Teil über die gängige, symmetrische Verschlüsselung ist für das Verständnis asymmetrischer Verschlüsselungstechniken wichtig.

Der Vollständigkeit her soll noch erwähnt werden, dass es auch noch eine dritte Art der Verschlüsselung gibt: Die Hash-Funktionen, welche jedoch nicht Bestandteil dieser Arbeit sind.[2]

1 Siehe 1. Anhang Tux
2 Siehe 2. Anhang Hash

2.1 **Die Mathematik der symmetrischen Verschlüsselung**

2.1.1 **Ein geschichtlicher Rückblick: Die Cäsarverschlüsselung**

Bereits fast 100 Jahre v. Chr. soll Julius Cäsar symmetrische Kryptographie verwendet haben, um geheime Nachrichten zu übermitteln, die keiner seiner Feinde entschlüsseln sollte. In diesem Kapitel werde ich einige Grundlagen der Kryptologie anhand der sogenannten Cäsarverschlüsselung erläutern. Um dieses Beispiel zu veranschaulichen, übernimmt Tux die Rolle des Imperators Cäsar und Tim wird sein Informant sein, der dem Imperator Tux geheime Botschaften zukommen lässt. Außerdem wurde das Beispiel für die heutige Anwendung im Bereich der Informationstechnologie und Mathematik aufgearbeitet.

2.1.2 **Gegenwärtige Umsetzung**

Um das Beispiel zu vereinfachen, beschränken wir uns auf das Alphabet der Kleinbuchstaben. Jedem dieser Buchstaben wird nun eine Zahl zugeordnet.
$a = 1$; $b = 2$; … $z = 26$;

Zusätzlich gilt:
M ist die zu verschlüsselnde Nachricht (engl. Message), auch Klartext (engl. Plaintext) genannt.
C ist die verschlüsselte Nachricht (engl. cryptographically secured).

$$M \in [1;26] \qquad C \in [1;26]$$

Damit es später für Computer keine Rechenschwierigkeiten, Probleme, oder Rundungsfehler aufgrund von Nachkommastellen gibt, sollen alle neu eingeführten Zahlen aus der Menge der nicht negativen Integer[3] sein.

$$Nicht-Negativer-Integer = \mathbb{N} = \{0,1,2,3,...\}$$

Wenn nun der Imperator Tux mit seinem Informanten Tim eine geheime Botschaft austauschen wollte, mussten sich die beiden erst einmal treffen. Bei diesem Treffen vereinbarten sie einen geheimen Schlüssel *k* (engl. key). Der

3 Siehe 3. Anhang Interger

kleinste Wert von *k* ist 0, der sinnvollerweise größte Wert für *k* ist die Anzahl der Buchstaben des verwendeten Alphabets – 1.

$$k \in [0; 25]$$

In unserem Beispiel wählten Tux und Tim für $k = 3$.

Wenn Tim nun Tux vertrauliche Informationen zukommen lassen will, verschlüsselt er diese wie folgt: Er schreibt sich das definierte Alphabet auf und darunter die entsprechenden Zahlen der Buchstaben.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Tabelle 1

Nun verschiebt er das ganze Alphabet um *k* Stellen und schreibt es unter das erste.

d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Tabelle 2, kodiert

Die Tabellen zeigen anschaulich, wie die Verschlüsselung funktioniert. Wenn nun der Buchstabe *a* kodiert werden soll, wird dieser einfach um 3 Stellen nach rechts verschoben und man enthält den Buchstaben *d*. Mathematisch heißt das:

$$C = M + k$$

Die Information wird nun zum Imperator Tux übermittelt. Dieser kann mit dem Wissen über *k* die Nachricht einfach wieder dekodieren, indem er die Verschlüsselungsfunktion umkehrt und nach *M* auflöst:

$$M = C - k$$

Doch Tim und Tux haben noch ein Problem: Tim will den Buchstaben *y* übermitteln.
Die graphische Darstellung zeigt, dass *y* nach der Kodierung ein *b* sein muss, doch mathematisch ist 25 + 3 = 28 und nicht 2. Aber die Tabellen verraten die Lösung: Nachdem Tim 25 + 1 gerechnet hat, ist er am Ende des definierten Alphabets angekommen und er beginnt von neuem zu zählen, das heißt, nach 26 kommt wieder die Eins.

Also rechnet man nicht mit den absoluten Zahlen, sondern stets mit ihren Resten bei der Division durch 26. So ist gesichert, dass *M* und *C* nicht größer als 26 werden und Rechnungen wie 25 + 3 werden trotzdem möglich. Diese Art der Rechnung wird mathematisch auch Modulo genannt. Hierbei entsteht ein Restklassenring $\mathbb{Z}_{26}$ [4].

Damit lautet der endgültige Algorithmus den Tim zum Verschlüsseln verwendet:

$$C = (M + k) \mod 26$$ # genannt *Cäsar_Verschlüsseln*

Da dies eine symmetrische Verschlüsselung ist, kann der Imperator Tux wieder durch Umkehren M errechnen:

$$M = (C + x) \mod 26$$

x muss die Umkehrung zu *k* sein, welche einfach (26 – *k*) ist. Eingesetzt ergibt sich damit

$$M = (C + 26 - k) \mod 26$$

Man sieht, dass man die 26 aufgrund der Modulo-Rechnung auch weglassen kann, wodurch sich

$$M = (C - k) \mod 26$$ # genannt *Cäsar_Entschlüsseln*

ergibt. Dieses Ergebnis wird auch durch die Veranschaulichung an der Tabelle 1 und der aus ihr resultierenden kodierten Tabelle 2 bestätigt.

Außerdem zeigt sich durch Einsetzen von *M* aus *Cäsar_Entschlüsseln* in Tims Verschlüsselungsfunktion *Cäsar_Verschlüsseln*:

$$C = ([(C - k) \mod 26] + k) \mod 26$$

Da *k* mit $k \in [0;25]$ bereits ein Element der Restklassenrings $\mathbb{Z}_{26}$ ist, lässt es sich in die Klammer ziehen:

$$C = ([(C - k + k) \mod 26]) \mod 26$$

$$C = ([(C) \mod 26]) \mod 26$$

Weil die Rechnung modulo 26 bei einem Element welches bereits in dem Restklassenring $\mathbb{Z}_{26}$ – also modulo 26 – ist, das Ergebnis nicht weiter verändert ergibt sich:

$$C = ([(C) \mod 26])$$

4 Siehe 4. Anhang Modulo und Restklasse

Weil *C* bereits als Element des Restklassenrings $\mathbb{Z}_{26}$ definiert sein sollte, zeigt sich, dass

$C = C$

Da nun die anfangs einfache Verschiebung einer Tabelle mit Buchstaben auf einen mathematischen Restklassenring abstrahiert wurde, muss noch eine Veränderung vorgenommen werden, damit der letzte Schritt mathematisch korrekt ist. Weil wir bei der Erstellung der Tabelle trivialerweise bei Eins angefangen und bei 26 aufgehört haben zu zählen, haben wir eines nicht beachtet:

$26 \; mod \; 26 = 0$

Um dies zu korrigieren, muss $M \in [0;25]$ und $C \in [0;25]$ gelten, damit $M \in \mathbb{Z}_{26}$ und $C \in \mathbb{Z}_{26}$. Um dies auch in der obigen Tabelle 1 umzusetzen, muss nur 26 durch 0 ersetzt werden, denn 0 ist relativ betrachtet nun der größte Wert. Dies gilt analog auch für Tabelle 2.[5] Damit ist der Beweis abgeschlossen und es gilt:

Kodieren: $C = (M + k) \quad mod \; 26$ # *Cäsar_Verschlüsseln*

Dekodieren: $M = (C - k) \quad mod \; 26$ # *Cäsar_Entschlüsseln*

2.1.3 **Allgemeine Definition**

Generell wird diese Art der Verschlüsselung **symmetrisch** genannt, denn beide Partner haben den gleichen Schlüssel *k* und der Algorithmus zum Entschlüsseln ist die Umkehrung der Verschlüsselung. Beide Partner spielen also sicherheitstechnisch eine symmetrische Rolle.

Allgemein gilt für jeden symmetrischen Verschlüsselungsalgorithmus:

Wenn ein Klartextbuchstabe *M* mit dem Algorithmus f und dem Schlüssel *k* verschlüsselt wird, kann er mit der Umkehrung des Algorithmus f^{-1} und dem Schlüssel *k* wieder entschlüsselt werden:

$f^{-1}(f(M)) = M$

Dies gilt nur, wenn für f und f^{-1} der gleiche Schlüssel *k* verwendet wird. Da die Kodierung so eindeutig wieder dekodiert werden kann, ergibt sich außerdem,

5 Siehe 5. Anhang Korrigierte Verschiebungstabellen

dass die Anzahl der Klartexte *M* nicht größer als die Anzahl der Geheimtexte *C* sein kann.

$|M| \leq |C|$ [6]

2.1.4 **Die Schwächen dieses Verfahrens**

Anhand dieses Verfahrens zeigen sich sofort einige Schwächen, die speziell die Cäsarverschlüsselung aufweist. Jedoch sind dies auch grundlegende Probleme, die bei symmetrischer Verschlüsselung auftreten können. Generell lässt sich leicht erkennen, dass ein Schlüssel, welcher nur aus einer Zahl zwischen 0 und 25 besteht, keine ausreichende Sicherheit bietet. Durch einen Brute-Force-Angriff [7] hätte man die entsprechende Zahl zu schnell erraten und könnte damit jede Nachricht entschlüsseln, denn die Anzahl der möglichen *k* ist sehr gering.
Auch bleibt das Verhältnis, wie oft bestimmte Buchstaben in bestimmten Sprachen auftreten, unverändert. Beispielsweise sind *e* und *n* die beiden häufigsten Buchstaben in deutschen Texten, folglich wären bei cäserverschlüsselten Texten (für k=3) *h* und *q* die zwei häufigsten Buchstaben, wodurch sich der Text auch ohne Wissen über *k* schnell entschlüsseln lässt.
Ein generelles „Problem“ ist das logische Problem des Schlüsselaustauschs bei symmetrischen Verschlüsselungsverfahren. Die beiden Kommunikationspartner Tux und Tim konnten dies zu Cäsars Zeiten noch als „Feature“ [8] nutzen, da sie sich damals noch persönlich trafen und den Schlüssel austauschten und somit ihre Authentizität sicherstellten. Doch wenn Tim heute das Internet besucht, kommuniziert er dort mit tausenden Tux, jedoch ohne einen von ihnen jemals getroffen zu haben. Folglich kennt weder Tim einen Schlüssel von Tux noch umgekehrt. Dies bedeutet, dass ein rein symmetrisches Verfahren zur sicheren Kommunikation zwischen Tim und Tux nicht mehr ausreicht.

Man könnte sich nun fragen, warum die beiden nicht einfach unverschlüsselt miteinander kommunizieren, was leider auch viel zu oft der Fall ist.

---

6 Quelle: Beutelspacher, Albrecht – Kryptologie – 7. Auflage; Seite 47
7 Siehe 6. Anhang Brute-Force
8 Siehe 7. Anhang Feature

2.1.5 **Das Problem „Internet“**

2.1.5.1 **Die Struktur des Internets**

Bereits im Wort Internet steckt das englische Wort „net“, welches sich treffend mit Netz übersetzen lässt. Folglich ist jeder Computer, der mit dem Internet verbunden ist, ein Teil dieses globalen Netzes. Im Folgenden wird jede Aktivität die Tim – ein Anwender im Netz – durchführt auf das Versenden und Empfangen von Datenpaketen beschränkt. Egal ob er E-Mails liest oder im Netz surft, alle empfangenen und versendeten Daten werden auf dieselbe Weise zugestellt: Sie werden in Pakete verpackt und an ihren Bestimmungsort geleitet. Da aber das Internet wie ein dezentrales Netz aufgebaut ist, werden Tims Pakete – aus denen sich all seine Aktivitäten, Passwörter und privaten Daten im Internet auslesen lassen – nur über Umwege zugestellt.

2.1.5.2 **Wer hat alles Zugriff auf Tims Pakete?**

Theoretisch könnte sich jeder mit entsprechender Kenntnis zufällig oder absichtlich in eine Position versetzen, um Zugriff auf Tims Pakete zu erlangen. Offensichtlich ist, dass alle Stationen des Netzes über die Tims Pakete geleitet werden – im folgenden Hops [9] genannt – direkten Zugriff auf Tims Datenpakete haben.

Doch auch die Verbindungen zwischen den einzelnen Hops sind nicht sicher und unterliegen nicht Tims Kontrolle. Tatsächlich ist es sowohl für den einfachen Anwender als auch für erfahrene Systemadministratoren normalerweise unmöglich zu erkennen, ob zwischen zwei Hops ein sogenannter Network-Sniffer [10] zwischengeschaltet wurde. Ein Network-Sniffer kann eine Kopie aller Pakete, die an ihm vorbei laufen, erstellen und sie beispielsweise automatisiert gezielt nach Passwörtern, Kontodaten oder weiteren sensiblen Informationen durchsuchen. Dabei werden die Pakete weder verändert noch auf sonstige Weise beeinflusst. Der Network-Sniffer ist somit unsichtbar.

Aufgrund dieser Sachverhalte ist die einzig vernünftige Art, sensible Daten im Internet zu übertragen, eine direkte Verschlüsselung der Verbindung zwischen Tim und Tux. An der Tatsache, dass alle Hops und weitere potentielle Angreifer

9 Siehe 8. Anhang Hop
10 Siehe 9. Anhang Network-Sniffer

(beispielsweise mit Netzwork-Sniffern) die Pakete mitlesen können, lässt sich damit nichts ändern, doch nur Tim und Tux können diese Pakete entschlüsseln, alle anderen lesen „Datenmüll“ mit.
Im Folgenden werde ich aufzeigen, wie die beiden dies realisieren können, obwohl sie sich zuvor noch nie persönlich begegnet sind.

2.2 **Die Mathematik der asymmetrischen Verschlüsselung**

Das Schlüsselwort hierfür lautet asymmetrische Verschlüsselung, oft auch Public-Key-Cryptography genannt.

2.2.1 **Theorie der asymmetrischen Verschlüsselung**

Die Lösung dieses in 2.1.4[11] beschriebenen Problems klingt erstaunlich einfach. Angenommen Tim und Tux haben sich identifiziert und reden direkt miteinander (ohne einen MitM[12]). Tux schickt Tim Anweisungen wie Tim seine Daten verschlüsseln soll. Durch die **Trennung von Verschlüsselung und Entschlüsselung** kann Tim mit seinem Wissen **nur noch Verschlüsseln** und es erfolgt somit automatisch eine Trennung in einen **privaten** und einen **öffentlichen Schlüssel** (daher der englische Name Public-Key-Cryptography). Tim kann nun seine Pakete an Tux schicken und ist sich sicher, dass niemand außer Tux sie wieder entschlüsseln kann.
Genauer gesagt: Wenn Tim etwas verschlüsselt, können **er und alle Angreifer** es nicht wieder entschlüsseln, dies kann dann nur noch Tux, da nur er den entsprechenden privaten Schlüssel besitzt. Jedoch kann jeder Angreifer selbst etwas verschlüsseln, dies bringt ihnen jedoch keinen Vorteil.
Man kann sich dieses Prinzip wie eine offene Haustür vorstellen. Jeder kann diese Türe schließen, danach ist sie ins Schloss gefallen und kann von niemanden mehr geöffnet werden, es sei denn er besitzt den Hausschlüssel (hier privater Schlüssel genannt). Tim legt also seine Daten in das Haus und lässt die Tür ins Schloss fallen. Das Haus wird kompakt verpackt und als Paket zu Tux geschickt. Da Tux als einziger den Hausschlüssel besitzt, kann er die Türe öffnen.

11 Siehe Seite 7
12 Siehe 10. Anhang Man in the Middle

2.2.2 **Allgemeine Definition**

Um dieses Prinzip mathematisch auszudrücken, führen wir nun einige neue Werte ein.

Der bisher eine allmächtige Schlüssel *k* wird ersetzt durch zwei zweckgebundene Schlüssel:

> *e* (engl. to **encrypt** = verschlüsseln) ist der **Public-Key**. *e* wird nur zum Verschlüsseln verwendet.
>
> *d* (engl. to **decrypt** = entschlüsseln) ist der **Private-Key**. *d* kennt nur Tux. Nur mit *d* kann entschlüsselt werden.

Der Verschlüsselungsalgorithmus *f* wird ersetzt:

> f_e ist die öffentliche, von *e* abhängige Funktion zum Verschlüsseln.
>
> f_d ist die von *d* abhängige Funktion zum Entschlüsseln.

e, f_e und f_d sind der Öffentlichkeit bekannt, jedoch kann Niemand außer Tux f_d ausführen, da dazu der Private-Key *d* benötigt wird. Folglich gilt für jedes asymmetrische Verschlüsselungssystem:

$$C = f_e(M)$$

$$M = f_d(C) \;\rightarrow\; M = f_d(f_e(M))$$

Damit der Public-Key *e* bedenkenlos veröffentlicht werden kann, muss er folgende Voraussetzungen erfüllen:

- Mit ihm darf nur verschlüsselt werden, zum Entschlüsseln wird der Private-Key benötigt.
- Aus dem Public-Key darf sich nicht der Private-Key berechnen lassen.
- Mit den Public-Key verschlüsselte Daten können nur von dem rechtmäßigen Empfänger – dem Besitzer des Private-Key – entschlüsselt werden

Damit dieses Prinzip nun praktisch am RSA-Algorithmus angewendet werden kann, sind zuerst einige zahlentheoretische Grundlagen nötig.

2.2.3 **Zahlentheoretische Grundlagen**

2.2.3.1 **Der euklidische Algorithmus**

Der griechische Mathematiker Euklid von Alexandria (ca. 350 v. Chr. bis ca. 300 v. Chr.)[13] stellt mit seinem nach ihm benannten Algorithmus die Grundlage für alle folgenden Rechnungen und Überlegungen zur Verfügung.

Funktionsweise des euklidischen Algorithmus:

Sind *a* und *b* zwei positive ganze Zahl und ist $a > b$, so berechnet der euklidische Algorithmus den *ggT(a,b)*. Obwohl dieser Algorithmus über 2000 Jahre alt ist, kann er besonders leicht systematisch von Computern berechnet werden. Die zeigt eine schematische Darstellung in einer Pseudo-Programmiersprache:

```
Euklidischer_Algorithmus  (a > b,   a∈ℕ\0   ,   b∈ℕ\0   ){
 Rest = mod(a,b)
 Solange ( Rest   ≠   0 ){
        a := b          # a wird ein neuer Wert zugewiesen, der alte Wert
                        # von a wird verworfen
        b := Rest
        Rest := mod(a,b)
 } Wenn ( Rest = 0 ){
       Ergebnis :=  b
 }
}
```

Erklärung:

Es muss nur eine Schleife so lange berechnet werden, bis die Abbruchbedingung „*Rest* ≠ 0“ nicht erfüllt ist und damit „*Rest* = 0“ gelten muss.

Als erstes wird der Rest bei der Division von *a* durch *b* berechnet:

$$Rest = mod(a, b)$$

Die zu erzielende Abbruchbedingung lautet, dass *b* ein Teiler von *a* ist, das heißt wie bereits erwähnt: „*Rest* = 0“

Da dieser Fall nur eintritt, wenn *mod*(*a,b*) = 0, werden folgende Schritte ausgeführt bis die Abbruchbedingung eintritt:

13 Quelle: http://de.wikipedia.org/wiki/Euklid

b wird vom Divisor zum Dividend: *a* := *b*

Der Rest wird zum Divisor: *b* := *Rest*

Nun wird wieder ein neuer Rest bestimmt: *Rest* := *mod*(*a,b*)

Damit beginnt die Schleife von neuem, mit dem neuen *Rest*, so lange bis eine Zahl gefunden ist die *a* und *b* restlos teilt.

Diese Zahl, hier *Ergebnis* genannt, ist der *ggT*(*a,b*)

Besondere Fälle:

Wenn $a = b \rightarrow ggT(a,b) = ggT(a,a) = a$

Wenn $a < b \rightarrow b > a$ werden die Variablennamen vertauscht und es gilt: $a > b$

2.2.3.2 **Die eulersche Phi-Funktion**

Die nach dem Mathematiker Leonhard Euler benannte eulersche Phi Funktion (geschrieben φ , oder ϕ bzw. Φ) ist eine zahlentheoretische Funktion, welche für jede positive natürliche Zahl *N* die Anzahl *x* der positiven, natürlichen Zahlen aus der Menge 1,2,...N zurück liefert, die teilerfremd[14] zu *N* sind. Man schreibt

$$x = \varphi(N)$$

Folglich gilt:

Der kleinstmögliche Wert für *x* ist Eins, da jede Zahl mindestens durch Eins teilbar ist. Der größtmögliche Wert für *x* ist *N,* wenn jede Zahl 1,2, ..., *N* teilerfremd zu *N* ist. Da $\varphi(1) = 1$ und $x \leq N$ gilt:

$$1 \leq x \leq N$$

Beispiel mit *N* = 5:

$$\varphi(5) = 4$$

Denn bei einem schematischen Probedurchlauf ergibt sich für *x* = 0 als Startwert

ggT(1,5) = 1	*x* = *x*+1 = 1
ggT(2,5) = 1	*x* = *x*+1 = 2
ggT(3,5) = 1	*x* = *x*+1 = 3
ggT(4,5) = 1	*x* = *x*+1 = 4
ggT(5,5) = 5	*x* = *x*+0 = 4

14 Siehe 11. Anhang Teilerfremd

die Bedingung $ggT(a,N) = 1$ für $1 \le a \le N$ ist erfüllt für folgende Werte von a:

$1,2,3,4 \rightarrow x = |a| = 4$

Damit ist $\varphi(N) = \varphi(5) = x = 4$

Aufgrund des Beispiels nehmen wir an: Für jede Primzahl p gilt:

$\varphi(p) = p - 1$

Diese Annahme lässt sich leicht bestätigen, da eine positive ganze Zahl p ($p \ge 2$) genau dann eine Primzahl ist, wenn sie nur durch 1 und p teilbar ist, ist p zu jeder Zahl 1,2, ... , (p-1) teilerfremd.

Da die Phi-Funktion laut Definition multiplikativ ist, gilt für eine weitere Primzahl q:

$\varphi(pq) = (p-1)(q-1)$

Diese einfache Übernahme aus der Definition lässt sich belegen, indem man das Ergebnis umformt:

$$\varphi(pq) = (p-1)(q-1) = pq - p - q + 1 = (pq-1) - (q-1) - (p-1) = pq - [1 + (q-1) + (p-1)]$$

Durch Überlegung zeigt sich, dass dieses Ergebnis der letzten Zeile richtig sein muss, denn pq währen alle Zahlen zwischen 1 und pq, ob teilerfremd oder nicht. Alle nicht zu pq teilerfremden Zahlen werden nun in der eckigen Klammer abgezogen:

1 zieht das nicht teilerfremde Produkt pq zu pq ab.

(q-1) und (p-1) ist die Anzahl aller restlichen nicht teilerfremden Zahlen zu pq da beide Zahlen Primzahlen sind. Folglich bleiben nur noch zu pq teilerfremde Zahlen übrig.

Damit gilt:

$(p-1)(q-1) = pq - [1 + (q-1) + (p-1)] = \varphi(pq)$ [15]

15 vgl.: Beutelspacher, Albrecht – Kryptologie – 7. Auflage; Seite 102

Diese Funktion bietet die Grundlage für den

2.2.3.3 **Der Satz von Euler-Fermat**

Der uns bereits bekannte Mathematiker Leonhard Euler stellte auf Grundlagen der Forschung des französischen Mathematikers Pierre de Fermat den nach ihnen benannten Satz von Euler-Fermat auf. Dieser besagt: Sind *M*, *N* zwei positive, ganze, teilerfremde Zahlen mit

$ggT(M,N) = 1$

so gilt:

$$M^{\varphi(N)} \mod N = 1$$

Oder geschrieben aus der Sicht der Restklasse modulo N ($\mathbb{Z}_N$):

$$M^{\varphi(N)} \equiv 1 \mod N$$ [16]

Man spricht: „M hoch Phi von N ist Eins kongruent Modulo N“

Da wir uns nun in einer Restklasse befinden, ist es nützlich, eine weitere Definition für das Rechnen in Restklassen kennenzulernen.

2.2.3.4 **Das Multiplikative Inverse**

> *„Ist n eine positive ganze Zahl und e eine zu n teilerfremde ganze Zahl, so gibt es eine ganze Zahl d mit* $0 \le d \le n-1$ *, sodass*
> $$d * e \equiv 1 \mod n$$
> *d wird auch als die 'sogenannte Multiplikative Inverse zu e modulo n' [...] bezeichnet.“* [17]

Da es beispielsweise für Rechnungen wie

$$\frac{3}{a} \equiv 1 \mod 7$$

generell keinen Lösungsansatz in der Restklasse $\mathbb{Z}_7$ gibt, da die Division nicht definiert ist, löst man dieses Problem über die Multiplikation. Mit anderen Worten ist eine Division in einer Restklasse eine Multiplikation mit dem Multiplikativen Inversen. So erhält man:

$$3 * b \equiv 1 \mod 7$$

16 Quelle: http://yimin.sinuslab.net/doc/mathematik_von_rsa.pdf; Seite 14

17 Quelle: http://yimin.sinuslab.net/doc/mathematik_von_rsa.pdf; Seite 12; Variablennamen geändert

Da 3 * 5 = 15 und 15 mod 7 = 1 kann man erkennen, dass das Multiplikative Inverse zu 3 in $\mathbb{Z}_7$ gleich $b = 5$ sein kann.
Für größere Zahlen lässt sich das Multiplikative Inverse einfach über den erweiterten euklidischen Algorithmus[18] ausrechnen, denn um auf die Definition des multiplikativen Inversen zurückzukommen: *e* und *n* sind teilerfremd, daraus folgt:

$$(d * e) \mod n = ggT(e, n) = 1$$

$$ggT(e, n) = e * d + n * y$$

Wenn man den Variablennamen von *d* auf *x* ändert erkennt man sofort, dass der erweiterte euklidische Algorithmus das Ergebnis für *x* und *y* liefert.
Diese zahlentheoretischen Grundlagen finden nun direkt ihre Anwendung im RSA-Algorithmus.

2.2.4 **Der RSA-Algorithmus**

Der RSA Algorithmus entstand 1977 als die drei Mathematiker Ronald L. **R**ivest, Adi **S**hamir und Leonard **A**dleman zu beweisen versuchten, dass eine asymmetrische Verschlüsselung unmöglich sei. Hierbei stießen sie auf einen Algorithmus, mit dem es doch möglich ist, eine asymmetrische Verschlüsselung durchzuführen. Dies gelang ihnen, als sie erkannten, dass die Faktorisierung einer sehr großen Zahl in ihre primen Bestandteile ein nahezu unlösbares Problem darstellt, umgekehrt jedoch die Erzeugung einer großen Zahl durch Multiplikation von Primzahlen nahezu keinen Rechenaufwand bedeutet.
Die Anwendung der RSA-Verschlüsselung beginnt mit der Erzeugung zweier Schlüssel: Einem geheimen, dem Private-Key (*d*) und einem öffentlichen Schlüssel, dem Public-Key (*e*).

18 Siehe 12. Anhang Erweiterter euklidischer Algorithmus

2.2.4.1 **RSA-Schlüsselerzeugung**

Damit nun die Voraussetzungen für eine Public-Key-Cryptography erfüllt sind[19], wird die Schlüsselerzeugung im RSA-Algorithmus durch sogenannte Falltürfunktionen verwirklicht. Der Name Falltürfunktionen verrät schon, dass sich Falltürfunktionen ohne Probleme in die eine Richtung berechnen lassen, aber rückwärts lassen sie sich nicht – oder nur mit Zusatzinformationen – berechnen. Wie bereits in der Einleitung erwähnt, handelt es sich dabei in RSA konkret um die Faktorisierung einer großen Zahl. Beim RSA-Algorithmus sind zwei Primzahlen die geheimen Zusatzinformationen aus denen sich die beiden Schlüssel berechnen lassen, aber ohne Kenntnis der Primzahlen sollte dies aufgrund des Faktorisierungsproblems nur unter sehr großem Rechenaufwand möglich sein.

Die zwei Primzahlen für die Erzeugung der Schlüssel werden p und q genannt. Sie werden durch Zufall von Tux erraten, und für sie gilt

$$p \neq q$$

Als zweites wird das sogenante RSA-Modul (genannt N) berechnet:

$$N = pq$$

Als nächstes wird im Schlüsselerzeugungsalgorithmus die eulersche Phi-Funktion von N berechnet.

$$\varphi(N)$$

Da N das Produkt der Primzahlen p und q ist, kennen wir das Ergebnis bereits:

$$\varphi(N)=(p-1)(q-1)$$

Als vierten und fünften Schritt muss nun e berechnet werden. Anschließend wird aus e (in RSA auch Verschlüsselungsexponent genannt) der geheime Schlüssel d (in RSA auch Entschlüsselungsexponent genannt) berechnet. Um Unklarheiten vorzubeugen möchte ich betonen, dass es für Tux sehr wohl möglich und einfach ist – entgegen unseren Voraussetzungen – aus e doch d zu berechnen. Für Außenstehende stellt dies jedoch ein nahezu unüberwindbares Hindernis dar.

19 mit e darf nur verschlüsselt werden, zum Entschlüsseln wird d benötigt und aus e darf sich nicht d berechnen lassen, vgl. Seite 10

Diese Thematik werde ich im Punkt **Sicherheit von RSA** (2.2.4.4) weiter behandeln.

Es soll nun *e* so gewählt werden, dass es sowohl größer als 1 und kleiner als $\varphi(N)$ ist. Außerdem muss *e* zu $\varphi(N)$ teilerfremd sein.

$e*d \equiv 1 \quad mod\ \varphi(N)$

$e*d \equiv ggT(e,\varphi(N)) \quad mod\ \varphi(N)$ # da *e* und $\varphi(N)$ teilerfremd

Damit ist die Definition für das Multiplikative Inverse zu *e* modulo $\varphi(N)$ für *d* erfüllt. Diese Gleichung muss nur noch nach *d* aufgelöst werden.

Überlegung 1:

Da sich der Divisionsrest (bei der Division durch $\varphi(N)$) nicht ändert wenn man k-mal den Divisor addiert oder subtrahiert ergibt sich

$e*d = 1-k\varphi(N)$ # Innerhalb der Modulorechnung

$e*d+k\varphi(N) = 1$

Diese Überlegung endet leider mit den zwei Unbekannten *d* und *k*. Deswegen ergibt sich folgende neue Überlegung:

Überlegung 2:

e ist teilerfremd zu $\varphi(N) \rightarrow ggT(e,\varphi(N)) = 1$

$ggT(e,\varphi(N)) = s*e+k*\varphi(N) = 1$

Diese neue Gleichung lässt sich mit dem erweiterten euklidischen Algorithmus errechen. Hier zeigt sich schön, dass *s* das multiplikative Inverse zu modulo $\varphi(N)$ ist $\rightarrow$

$s=d$

$ggT(e,\varphi(N)) = d*e+k*\varphi(N)$

Dies bestätigt nun auch die erste Überlegung, denn nun zeigt sich, dass diese Gleichung sich auch mit dem erweiterten euklidischen Algorithmus lösen ließe und das Ergebnis mit der zweiten Überlegung übereinstimmen würde.

Der Public-Key wird nun aus dem Zahlenpaar (*e*,*N*) gebildet, der Private-Key aus dem Zahlenpaar (*d*,*N*).

p und *q* sollten nun sicher vernichtet werden, damit die Integrität der Schlüssel gesichert ist.

2.2.4.2 **Der Verschlüsselungsvorgang**

Kodieren: $C \equiv M^e \mod N$ # genannt *RSA_Verschlüsseln*

Dekodieren: $M \equiv C^d \mod N$ # genannt *RSA_Entschlüsseln*

Es zeigt sich:

$M \equiv C^d \mod N$ # Einsetzen von *RSA_Verschlüsseln* in
RSA_Entschlüsseln

$M \equiv M^{ed} \mod N$ # (*ed*) auflösen

Da bei der Erzeugung von *e* und *d* galt:

$e*d \equiv 1 \mod \varphi(N)$

$(e*d) \mod \varphi(N) = 1$

Durch anschauliches Auflösung des Modulo in eine Division ergibt

sich:

$\frac{e*d}{\varphi(N)} = k\varphi(N) + 1$

$M \equiv M^{1+k\varphi(N)} \mod N$ # Einsetzen in die Restklasse modulo N [20]

$M \equiv M*M^{k\varphi(N)} \mod N$ # Satz von Euler-Fermat *k*-mal anwenden

$M \equiv M*1 \mod N$

$M \equiv M \mod N$

Dies beweist, dass die Verschlüsselung funktioniert.

Weiteres Anwendungsgebiet

Eine andere Anwendung von RSA ist – im Gegensatz zu der Verschlüsselung von Geheimtexten – das Signieren von öffentlichen Dokumenten um ihre Integrität zu bestätigen. Prinzipiell liegt der Signatur der selbe Algorithmus wie bei der Verschlüsselung zu Grunde, jedoch wird hier ein komplett gegensätzliches Ziel erreicht. Deshalb wird die Signatur von Dokumenten häufig auch als die Umkehrung der Verschlüsselung bezeichnet. Dementsprechend verschlüsselt Tim sein

20 Siehe 13. Anhang Lösung über den chinesischen Restsatz

öffentliches Dokument mit seinem geheimen Private-Key und veröffentlicht es mit seinem Public-Key. Dank des Public-Keys kann jeder das Dokument entschlüsseln. Ein Angreifer kann es jedoch nicht ändern, da er zum Wiederverschlüsseln Tims Private-Key benötigte.

2.2.4.2 **Beispiel**

Ein einfaches Beispiel soll nun die Anwendung des RSA-Algorithmus weiter veranschaulichen.

Tux erzeugt nun zuerst seinen Public- und Private-Key. In unserem Beispiel errät er die zwei Primzahlen anhand der Anzahl der Schneeflocken die auf seinem Fensterbrett liegen. Dies ist nur ein Beispiel für eine sehr sichere Möglichkeit Zufallszahlen zu erzeugen, denn man muss sich stets vor Augen halten, dass es gängigen Computern nicht möglich ist Zufälle zu erzeugen.

$p = 59$

$q = 151$

$N = p * q = 8909$

$\varphi(N) = (p-1)(q-1) = 8700$

Nun wählt Tux den Bestandteil des öffentlichen Schlüssels *e* teilerfremd zu $\varphi(N)$ anhand der Kaffeetassen auf seinem Schreibtisch.

$e = 11$

Da $ggT(e, \varphi(N)) = ggT(11,8700) = 1$ kann Tux nun *d* berechnen.

$ggT(11,8700) = d * 11 + k * 8700$

Nach 11 Tassen Kaffee löst auch Tux diese Gleichung schnell und es ergibt sich

$d = 791$

Nun vernichtet Tux die Zahlen *p* und *q* und spült die Kaffeetassen ab, damit niemand mehr seine Rechnung rückwärts rechnen kann.

Tux Public-Key ist damit (*11*,*8700*) und wird an Tim geschickt, den Private-Key (*791*,*8700*) bewahrt Tux sicher auf.

Nun ist es an der Zeit, dass Tim ein „Hallo !“ an Tux sendet. Dazu verwendet er seine Funktion *RSA_Verschlüsseln:*

$C \equiv M^e \mod N$ # *RSA_Verschlüsseln* von Tim

Tim übersetzt nun die Buchstaben in Zahlen. Da sein Text „Hallo !“ weit über das von uns anfangs in Tabelle 1 und 2 definierte Alphabet aus nur 26 Großbuchstaben hinaus geht, verwendet er dazu den international anerkannten Zeichensatz UTF-8. Auch hier muss Tim nur in einer Tabelle aus Buchstaben nachschlagen um die entsprechenden Zahlen zu finden.[21]

H	72	
a	97	
l	108	
l	108	
o	111	
	32	# Leerzeichen
!	33	

Tim fängt an mit H

$$C \equiv 72^{11} \quad mod\ 8700$$

$$C = 6722$$

Diese geheime Zahl kann schon zu Tux geschickt werden und von ihm dekodiert werden:

$$M \equiv C^{d} \quad mod\ N \qquad \#\ RSA\_Entschlüsseln \text{ von Tux}$$

$$M \equiv 6722^{791} \quad mod\ 8700$$

$$M = 72$$

Da auch Tux als Zeichensatz UTF-8 verwendet, schlägt er 72 in der Zeichensatztabelle nach und findet dort ein großes H. Analog können nun Tim und Tux mit den restlichen Zeichen fortfahren.

2.2.4.4 **Sicherheit von RSA**

Der Public-Key enthält für einen Angreifer die Informationen *e* und *N*. Damit der Angreifer die gesuchte Information *d* errechnen kann, muss er folgende Gleichung mit dem erweiterten euklidischen Algorithmus lösen können:

$$ggT(e, \varphi(N)) = d*e + k*\varphi(N)$$

21 Siehe 14. Anhang UTF-8 Zeichentabelle

Dazu muss der Angreifer allerdings $\varphi(N)$ kennen. Jedoch ist die einzige Möglichkeit $\varphi(N)$ auszurechnen[22] die von uns bereits verwendete Gleichung $\varphi(N)=(p-1)(q-1)$

Hier zeigt sich, dass diese Gleichung – wie bereits erwähnt – nicht ohne das Wissen über *p* und *q* berechnet werden kann. Folglich muss ein Angreifer die natürliche Zahl *N* in die zwei Primzahlen *p* und *q* faktorisieren können. Die einzige Möglichkeit, die bis jetzt für diese Aufgabe bekannt ist, ist das systematische Durchprobieren aller in Frage kommenden Zahlen. Auch wenn es mathematische Algorithmen gibt, die das Faktorisieren großer Zahlen erleichtern, bleibt ein solcher Angriff immer noch ein Brute-Force Angriff, der durch die Verwendung immer größerer Primzahlen stark erschwert wird. Damit gilt RSA als mathematisch sicher, da es keine mathematischen Schwächen gibt, sondern einzig und allein durch Brute-Force geknackt werden kann.
Beispielsweise wurde 2005 eine 640bit[23] Zahl erfolgreich faktorisiert[24]. Dennoch gilt RSA deswegen nicht als unsicher denn:

1) Zu diesem Zeitpunkt bereits 1024bit Zahlen eingesetzt wurden und 2048bit empfohlen wurde.
2) Die für diese Aufgabe verwendeten Clustersysteme mit der entsprechenden Rechenleistung aufgrund der extrem hohen Anschaffungskosten nicht jedem zur Verfügung stehen.
3) Die Faktorisierung trotz der extrem schnellen, verwendeten Rechner immer noch fünf Monate Dauerbetrieb benötige.

Folglich bleibt RSA auch so lange sicher, bis es eine Lösung für das mathematische Problem der Faktorisierung großer Zahlen gibt oder Tim und Tux vergessen alle paar Jahre – mit der steigenden Rechenleistung handelsüblicher Computer – ihre Primzahlen zu erhöhen.

22 probieren, raten und Brute-Force sei an dieser Stelle ausgeschlossen, da dies bei großen Zahlen ein extrem rechenaufwändiger Prozess wäre
23 Siehe 15. Anhang Bit
24 Siehe 16. Anhang Heise-News

3 **Abschließende Bewertung und Ergebnisse**

Meiner Meinung nach veranschaulicht diese Arbeit gut, wie sich durch einfache mathematische Anwendungen starke Kryptosysteme bauen lassen. Dennoch beinhaltet diese Arbeit auch nur den ersten Baustein einer sicheren Kommunikation über das Internet. Der zweite – mindestens eben so wichtige Aspekt – ist die richtige Implementierung der Verschlüsselungsalgorithmen[25]. Nur eine Kombination aus mehreren Verschlüsselungsalgorithmen kann unangenehmen Überraschungen vorbeugen, wenn einer dieser Algorithmen plötzlich geknackt werden sollte. Zusätzlich sollte man stets die Vorteile der symmetrischen und asymmetrischen Kommunikation in sogenannten Hybridverfahren – wie dem gängigen SSL und TLS – nutzen. Ein asymmetrisches Verschlüsselungsverfahren ist perfekt um Passwörter auszutauschen und die Authentizität zu prüfen. Leider ist es dementsprechend rechenintensiv und langsam, deswegen sollte sofort, nachdem Tim und Tux auf asymmetrischem Wege einen Schlüssel *k* getauscht haben, ein symmetrisches Verschlüsselungsverfahren mit dem geheimen Schlüssel *k* genutzt werden. So erhält man Kryptosysteme, die sowohl hochperformant und sicher arbeiten, als auch von ihren jeweiligen logischen Schwachstellen befreit wurden. Ein sicherer symmetrischer Kryptoalgorithmus ist etwa AES, welcher beispielsweise von amerikanischen Geheimdiensten genutzt wird, den ich aber auch für meine Festplatte verwende. Wie ich bereits erwähnt habe, ist die Kryptographie eine sehr einfache und logische Anwendung der Mathematik, deswegen passt auch der AES-Algorithmus in eine Zeile:

„$C_i = E_{K1}(P_i \text{ ^ } (K2 \otimes i)) \text{ ^ } (K2 \otimes i)$“ [26]

Eine ausführliche Erklärung und Analyse des Algorithmus hingegen nicht. Dennoch möchte ich erwähnen, dass die Kombination von RSA und AES noch heute genutzt wird um sicher über das Internet zu kommunizieren.

25 an dieser Stelle möchte ich noch einmal auf „10. Anhang Man in the Middle“ verweisen
26 Quelle: http://www.truecrypt.org/docs/modes-of-operation.php

Beispielsweise im ssl- (oder seinem Nachfolger tls-) Protokoll. Dieses Protokoll bildet auch die Grundlage für ssh[27] und https (http secure), eine gängige Variante um das unverschlüsselte http-Protokoll zu sichern. Https wird unter anderem genutzt für: eBay, Onlinebanking, Webmails und viele weitere Dienste. Dennoch sind alle diese Anwendungen nur sicher, wenn sie nicht nur mathematisch sicher sind, sondern wenn auch die Implementierung sicher ist.
Solche Implementierungsfehler sind beispielsweise schlechte Zufallsgeneratoren, ungenügender Schutz des gespeicherten Private Keys bzw. Festplattendiebstahl, ungenügender Schutz des Arbeitsspeichers, welcher alle geheimen Keys temporär speichern muss, damit der Computer damit rechnen kann, Anfälligkeit für Man in the Middle Angriffe, fehlende Prüfung der Authentizität, Unterstützung veralteter Protokolle, ungenügende Schlüssellänge, nicht abgeschlossene Serverräume und nicht vertrauenswürdige Mitarbeiter.
Trotz all diese Gefahren hat die Mathematik jedoch ihr Ziel erreicht. Keines diese Szenarien führt auf mathematischer Fehler zurück, sondern nur auf unsachgemäße oder grob fahrlässige Anwendung durch den Benutzer.

27 Der ssh-Verbindungsaufbau stellt auch das Titelbild dieser Facharbeit bereit. Das Titelbild des Anhangs zeigt die sichere Implementierung von RSA in ssh-Protokoll: Da ein MitM erfolgreich erkannt wurde, wurde der Verbindungsaufbau – d. h. der Austausch von *k* mit Hilfe von RSA – abgebrochen. Der „RSA key fingerprint“ ist in diesem Zusammenhang nichts anderes als ein Hash-Wert von Tux Publickey. Da sich dieser grundlos geändert hat, ist von einem MitM-Angriff auszugehen. Dies zeigt beeindruckend das lückenlose Zusammenspiel von symmetrischer, asymmetrischer Verschlüsselung und Hash-Algorithmen.

Quellennachweis

Bildnachweis:

Illustration „Tux“, Anhang Punkt 1:

Quelle: http://de.wikipedia.org/wiki/Bild:NewTux.svg

Copyright:

„The copyright holder of this file Larry Ewing allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.“

Alle weiteren verwendeten Bilder hab ich selbst erstellt.

Literatur:

Beutelspacher, Albrecht
Kryptologie
Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen
7., verbesserte Auflage
vieweg
ISBN 3-8348-0014-7

Witt, Kurt-Ulrich
Algebraische Grundlagen der Informatik
Strukturen - Zahlen - Verschlüsselung - Codierung
2. Auflage Januar 2005
vieweg
ISBN 3-528-13166-7

Internet:

Bedeutung der Primzahlen in der asymmetrischen Kryptographie am Beispiel von RSA
http://vbinside.de/facharbeit_dom/facharbeit_dominik.zip, 28.12.2007
Autor: Auras, Dominik
Veröffentlichung: 07.03.2003

RSA-640 geknackt
http://www.heise.de/newsticker/meldung/65957, 09.11.2005; 17:45
Herausgeber: cr/c't; Heise Zeitschriften Verlag

UTF-8-Codetabelle mit Unicode-Zeichen
http://www.utf8-zeichentabelle.de/, 28.12.2007
Inhaber der Webseite:Schild, Tomas

Modes of Operation
http://www.truecrypt.org/docs/modes-of-operation.php, 03.12.2007
Herausgeber: TrueCrypt Foundation

Internet
http://de.wikipedia.org/wiki/Internet, 06.02.2007
Herausgeber: Wikimedia Foundation

RSA-Kryptosystem
http://de.wikipedia.org/wiki/RSA-Kryptosystem, 10.02.2007
Herausgeber: Wikimedia Foundation

Eulersche φ-Funktion
http://de.wikipedia.org/wiki/Eulersche_%CF%86-Funktion, 14.03.2007
Herausgeber: Wikimedia Foundation

Teilerfremdheit
http://de.wikipedia.org/wiki/Teilerfremdheit, 14.03.2007
Herausgeber: Wikimedia Foundation

Erweiterter euklidischer Algorithmus
http://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus, 28.05.2007
Herausgeber: Wikimedia Foundation

Brute-Force-Methode
http://de.wikipedia.org/wiki/Brute_force, 20.08.2007
Herausgeber: Wikimedia Foundation

Satz von Euler
http://de.wikipedia.org/wiki/Satz_von_Euler, 27.08.2007
Herausgeber: Wikimedia Foundation

Euklid
http://de.wikipedia.org/wiki/Euklid, 28.10.2007
Herausgeber: Wikimedia Foundation

Restklasse
http://de.wikipedia.org/wiki/Restklasse, 28.10.2007
Herausgeber: Wikimedia Foundation

Restklassenring
http://de.wikipedia.org/wiki/Restklassenring, 28.10.2007
Herausgeber: Wikimedia Foundation

Die Mathematik von RSA
http://yimin.sinuslab.net/doc/mathematik_von_rsa.pdf, 19.02.2007
Autor: Yimin Ge
Veröffentlichung: August 2005

Software:

CrypTool
Binärdatei: CrypTool_1_4_10_de.exe
Verwendete Version: 1.4.10
Quelle: http://www.cryptool.de/
Auf der Webseite angegebener Maintainer und Ansprechpartner: Bernhard Esslinger

Ich erkläre hiermit, dass ich meine Facharbeit ohne fremde Hilfe angefertigt habe und nur die im Quellennachweis angeführten Quellen und Hilfsmittel benützt habe.

....., den .....

Heinrich-Heine-Gymnasium

Facharbeit

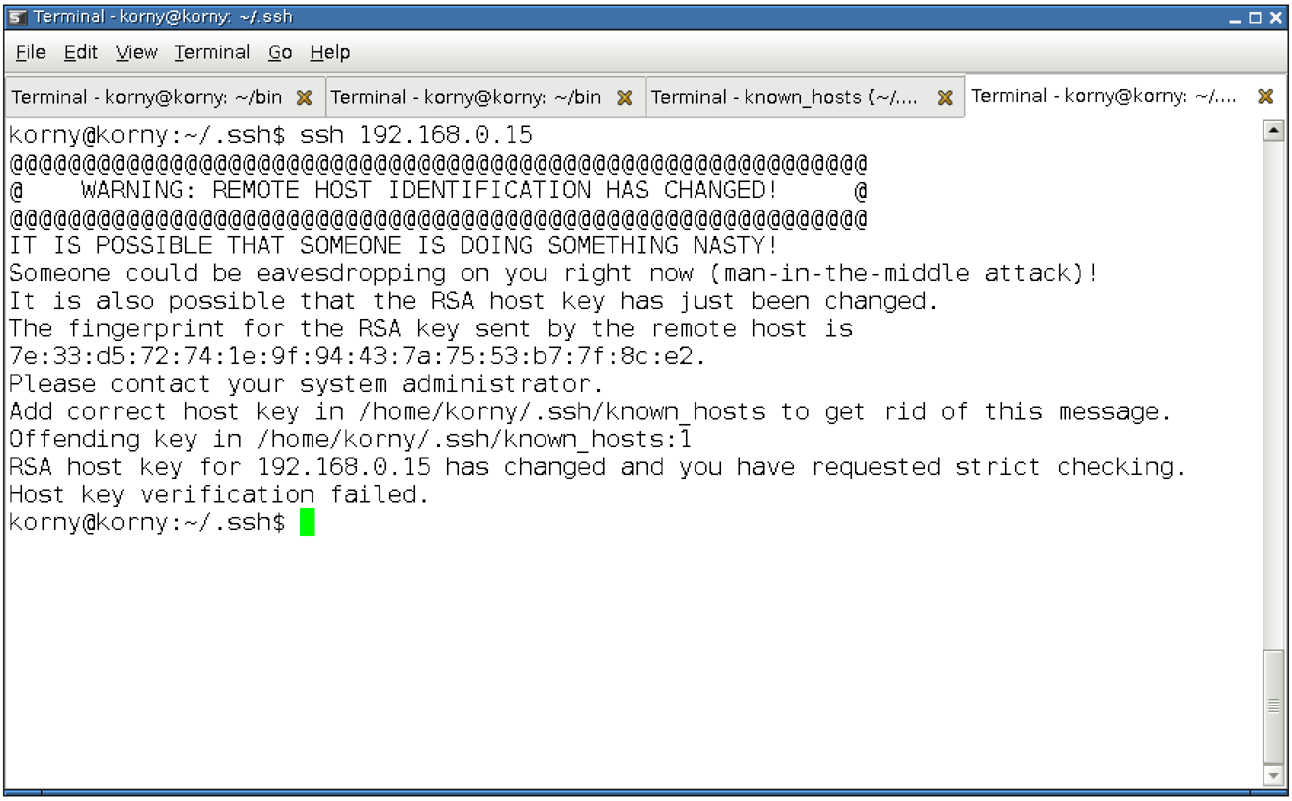
aus dem Fach

Mathematik

Thema:

Mathematik der Kryptographie

Anhang



Verfasser: Cornelius Diekmann

Kursleiter: Frau Nikles

1. Anhang Tux

Tux ist ein Pinguin und stellt das Maskottchen des freien Betriebssystems Linux dar.
Ich möchte Tux hier als Namen verwenden, da eine große Menge der Server im Internet auf Linux aufbauen.
Deswegen wird Tux in dieser Arbeit auch die Rolle eines Servers übernehmen und Tim sein Client sein, der die Dienste des Tux in Anspruch nimmt.
Außerdem repräsentiert ein Tux nicht nur einen einfachen Server, er vertritt auch die Prinzipien der freien und quellenoffenen Gemeinschaft. Dies ist besonders auf diese Facharbeit zu beziehen, da nur eine quellenoffene Verschlüsselung von unabhängigen Leuten geprüft werden kann und damit sicherstellt, dass diese Verschlüsselungstechnik sowohl mathematisch korrekt ist, als auch dass die Entwickler kein geheimes Hintertürchen eingebaut haben.

2. Anhang Hash

Hashfunktionen sind aufgrund der Seitenbegrenzung nicht Bestandteil dieser Arbeit, dennoch möchte ich einen kurzen Text über Rainbowtables – mit denen es möglich ist Hash-Werte zu knacken – beilegen:
Rainbowtables sind – wie der Name schon sagt – eine bunte Tabelle mit der es möglich ist, Hash-Werte zu entschlüsseln. Doch entschlüsseln ist eigentlich hier das falsche Wort. Ich fange am Anfang an, damit das Ausmaß der Bedrohung von Rainbowtables klar wird:
Wenn man sich mittels Benutzername und Passwort authentifiziert (z.B.: Windowskennwort eintippen, eBay und Foren einloggen, E-Mail abrufen ..), identifiziert man sich über die Kombination aus Benutzername und Passwort. Diese Daten – vor allem das Passwort – müssen sicher gespeichert werden, dazu nutzt man Hashfunktionen wie md5 oder sha. Diese Funktionen sind momentan eine der sichersten Möglichkeiten ein Passwort zu verschlüsseln, denn einmal verschlüsselt, kann man es nicht wieder entschlüsseln.

Hashfunktionen sind also mathematische Einwegfunktionen, d.h. eine Umkehrung (= Entschlüsselung) ist nicht möglich. Dennoch liefert die Hashfunktion von einem Wert immer den selben Hash zurück.
Der md5-Hash von 'passwort' ist beispielsweise immer:
„e22a63fb76874c99488435f26b117e37“

Dieser Wert – und nicht das Passwort im Klartext – wird in der Datenbank mit dem zugehörigen Namen gespeichert. Wenn der Benutzer sich nun einloggen will, berechnet der Computer aus dem Passwort, welches der Benutzer zum einloggen verwendet hat, den md5-Hash. Wenn dieser Hash mit dem aus der Datenbank übereinstimmt, hat sich der Benutzer authentifiziert und gilt als eingeloggt.

Wenn nun ein Angreifer die Hash-Werte aus der Datenbank oder der Festplatte des Computers klaut (dies ist die erste Hürde, die ein Angreifer überwinden muss), kann er also nicht das Passwort entschlüsseln. So weit so gut, doch der Angreifer hat noch eine Möglichkeit an die Passwörter zu kommen: Durchprobieren (ein Brute-Force Angriff).

Er berechnet den md5-Hash für 'a', welcher
„0cc175b9c0f1b6a831c399e269772661“ist.

Nun stellt der Angreifer fest, dass dieser Wert nicht mit dem von ihm erbeuteten übereinstimmt. Jetzt probiert der Angreifer solange Kombinationen aus, bis er ein Wort hat, welches den richtigen Hash-Wert liefert. Eine Wörterliste mit häufigen Passwörtern beschleunigt diesen Vorgang sehr, doch wenn das Opfer ein gutes Passwort hat, kommt der Angreifer nicht in einer angemessenen Zeit an das Passwort des Opfers. Der Angreifer könnte sich nun eine Datenbank mit jedem möglichen Passwort und dem dazugehörigen Hash anlegen, doch dies würde viel zu viel Speicherplatz belegen.
Und genau hier kommen Rainbowtables zum Einsatz. Sie sind nichts anderes als eine speicherplatzoptimierte Lösung mit der man – nach einmaliger Erzeugung der Tabelle – mit vergleichsweise wenig Rechenaufwand zum dem Hash-Wert das passende Passwort suchen kann. So kann man – wenn man sich einmal eine

Rainbowtable für einen Hash erstellt hat – sehr viele Hashes in kurzer Zeit entschlüsseln. Der aufzubringende Zeitaufwand ist natürlich die Erstellung der entsprechenden Rainbowtable. Je nach Umfang kann das bis ins Unendliche gehen. Und jede Rainbowtable funktioniert auch nur für einen bestimmten Hash-Algorithmus.
Der Nachteil der Rainbowtable Methode ist damit die Erstellung der Tabelle, da nur eine leichte Modifizierung eines Hash-Algorithmuses alle schon bestehenden Tabellen überflüssig macht und eine neue Tabelle erstellt werden muss. Auf dem Stand der Syss (Systems 2007) meinte ein Passwortknacker bei einer Vorführung, er habe vier Rechner eine ganze Woche lang durchrechnen lassen, um die Rainbowtable für einen Algorithmus zu erstellen, mit der es ihm nur möglich war ein Windowspasswort einer vorher festgelegten Länge, zu entschlüsseln.

3. Anhang Integer

Als Integer (engl. ganze Zahl) bezeichnet man in der Informatik einen Datentyp der eine ganze Zahl speichern kann.
Integer = {.., -2, -1, 0, 1, 2, .. }

4. Anhang Modulo und Restklasse

Modulo, oft abgekürzt mit mod, ist eine mathematische Funktion. Seien *a* und *b* zwei ganze Zahlen, dann ist *a* mod *b* der Rest der Division von *a* durch *b*. Es gilt: *b* teilt *a* genau dann, wenn es eine ganze Zahl *k* gibt, sodass gilt $a = k * b$, dann ist *a* mod *b* = 0, denn es gibt keinen Rest bei der Division. Wenn $a = k * b + x$, dann ist *a* mod *b* = *x*. Dies führt uns zur
Definition der Restklasse:

> *„Im mathematischen Teilgebiet der Zahlentheorie ist die **Restklasse** einer Zahl a modulo einer Zahl m die Menge aller Zahlen, die bei Division durch m denselben Rest lassen wie a.“* [1]

Beispielsweise ist:
15 mod 10 = 5
25 mod 10 = 5

1 Quelle: http://de.wikipedia.org/wiki/Restklasse

Damit sind 15 und 25 in der selben Restklasse modulo 10. Man schreibt auch

$15 \equiv 5 \mod 10$

$25 \equiv 5 \mod 10$

Man sagt 15 und 25 sind kongruent modulo in der Restklasse mod 10. Aus vielen einzelnen Restklassen lässt sich nun ein Restklassenring bilden.

Definition Restklassenring

> *„Ist $n \geq 2$ eine natürliche Zahl, dann werden ganze Zahlen mit gleichem Rest bei Division durch n zu sogenannten **Restklassen** modulo n zusammengefasst. Zwei ganze Zahlen sind also in derselben Restklasse, wenn ihre Differenz durch n teilbar ist. Die Restklassen bilden zusammen [...] den **Restklassenring**, der mit $\mathbb{Z}/n\mathbb{Z}$ oder $\mathbb{Z}/n$ oder $\mathbb{Z}_n$ bezeichnet wird (sprich 'Z modulo n').“* [2]

So besteht beispielsweise der Restklassenring $\mathbb{Z}_{10}$

x mod 10

aus den einzellnen Restklassen [0], [1], .. [9]

Die Restklasse [5] enthält dann beispielsweise unter anderem unsere bekannten Zahlen 15 und 25. Die Restklasse [0] enthält alle Zahlen die restlos durch 10 teilbar sind.

Umgangssprachlich wird ein Restklassenring auch oft Restklassengruppe genannt, da mehrere Restklassen zu einer Gruppe zusammengefasst werden.

5. Anhang Korrigierte Verschiebungstabellen

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0

Tabelle 1 korrigiert

d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0

Tabelle 2 korrigiert, kodiert

2 Quelle: http://de.wikipedia.org/wiki/Restklassenring

6. Anhang Brute-Force

In der Kryptologie versteht man unter der Brute-Force-Methode (übersetzt: Methode der rohen Gewalt) die Möglichkeit den geheimen Schlüssel *k* ohne Vorwissen über *k* zu erraten.
Dies geschieht durch reines Durchprobieren aller möglichen Schlüssel $\tilde{k}$, solange bis ein Schlüssel einen sinnvollen Klartext entschlüsselt. Damit ist mit großer Wahrscheinlichkeit $\tilde{k} = k$.

Brute-Force-Angriffe führen normalerweise nach einer bestimmten Zeit immer zum richtigen Ergebnis, doch man kann diese Angriffe vereiteln, indem man lange Schlüssel mit vielen verschieden Zeichen wählt. So sollte im Idealfall die Zeit bis ein Angreifer alle Schlüssel durchprobiert hat mindestens doppelt so lang sein, als wie die verschlüsselte Information schützenswert ist. Die Zeitangabe „mindestens doppelt so lang“ errechnet sich daraus, dass ein Angreifer im Durchschnitt bei der Hälfte aller möglichen Kombinationen für *k* einen passenden Schlüssel $\tilde{k} = k$ findet.

7. Anhang Feature

Feature (engl.) bedeutet übersetzt im Bereich der Informatik so viel wie Hauptbestandteil oder wichtiges Merkmal eines Produkts im positiven Sinne.
„*That's not a bug, it's a feature*“ ist ein bekannter Spruch, der häufig in der Informatik verwendet wird, um offensichtliche Fehler oder logische Schwachstellen in einer Software zu verschleiern.
"*BUGS*

Our software never has bugs.

It just develops random features. ;) " [3]

3 Zitat aus der manpage eins Programms. Aufgrund von aktuellen Unklarheiten im §202c StGB verzichte ich auf die namentliche Nennung dieser Sicherheitsanalysesoftware.

8. Anhang Hop

Vereinfacht bezeichnet man jeden Computer, vor dem nicht Sender oder Empfänger eines Paketes sitzen, aber über den dieses Paket bis zu seinem Bestimmungsort geleitet wird als Hop. Formal kann man einen Hop also auch als einen Netzwerkknoten beschreiben, dort wo alle Pakete zusammenlaufen und sich wieder trennen, um ihre Reise fortzusetzen.

Bildlich wäre es, wenn man sich ein Spinnennetz – welches das Internet symbolisiert – vorstellt. An einem Ende sitzt Tim, am anderen Tux. Sie schicken ihre Pakete über die Spinnenweben. Dabei passiert jedes Paket mehrere Knotenpunkte im Spinnennetz, um an sein Ziel zu kommen. Dies Knotenpunkte sind zu vergleichen mit Hops.

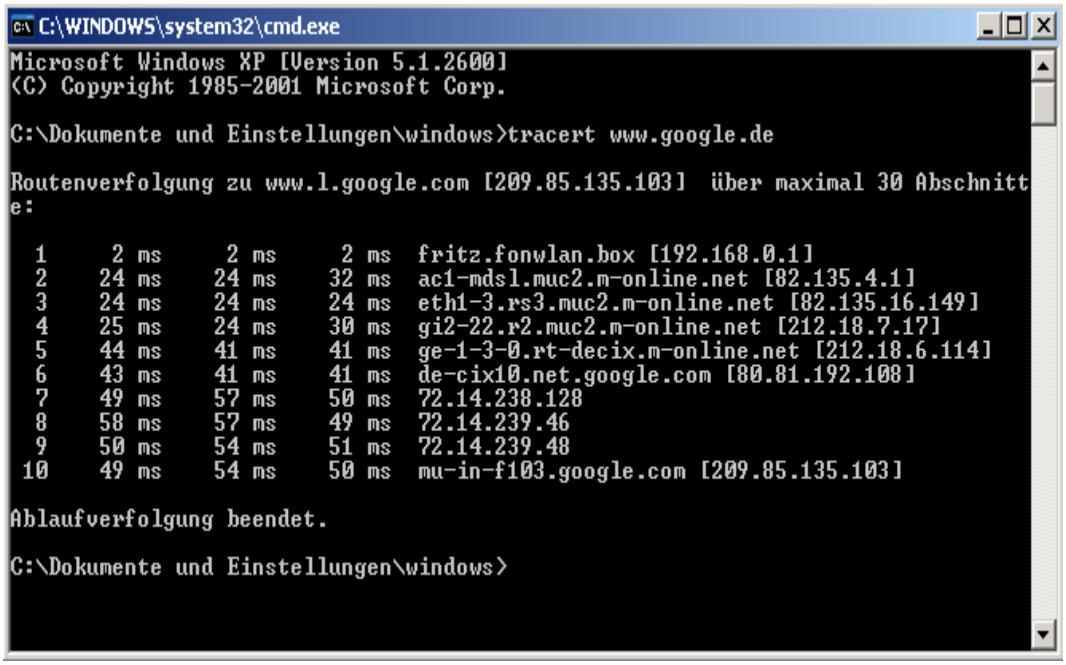
8.1 Anhang Hop Nachtrag

Bei einem Windowssystem lassen sich die Hops über den Befehl tracert herausfinden. Beispielweise:

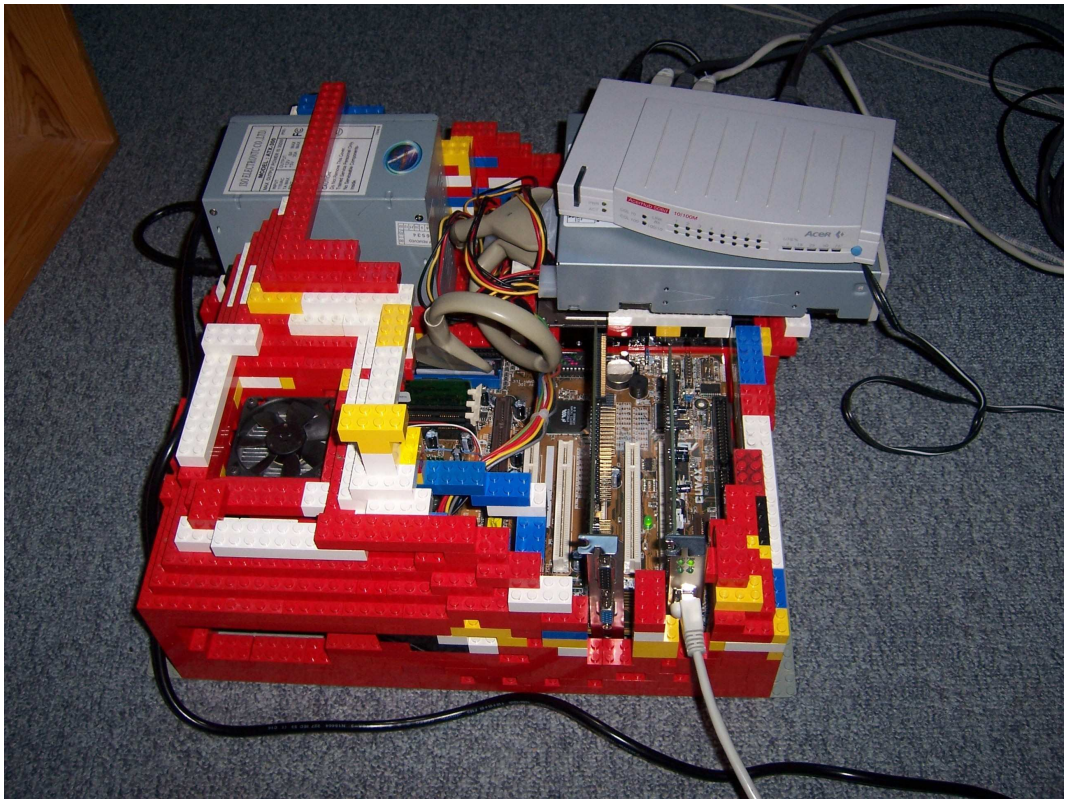
Strat -> Ausführen -> „cmd“ [Enter] -> „tracert www.google.de“ [Enter]

Unter Linux ist der Shellbefehl „traceroute www.google.de“ das entsprechende Äquivalent.

In meinem Beispiel sind das 10 Stationen über die meine Pakete geleitet werden, von denen ich 8 nicht kenne und ihnen somit auch nicht vertrauen kann.



9. Anhang Network-Sniffer



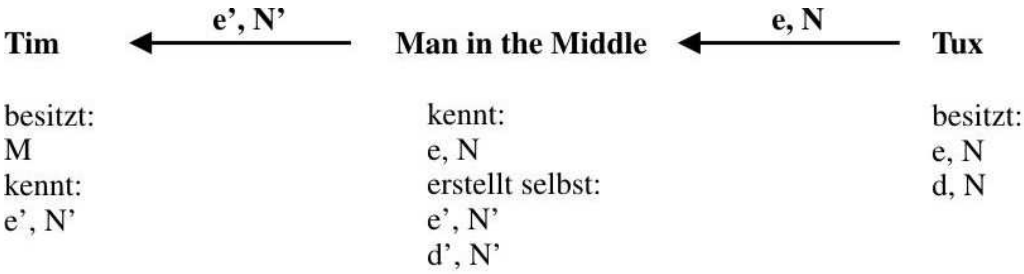
Ein von mir selbstgebauter Network-Sniffer aus Lego© und einem alten PC.

Dieses unscheinbar wirkende Konstrukt aus Lego© ist in der Lage jedes einzelne Paket, welches bei mir in der Wohnung versendet wird, zu lesen und abzuspeichern. Ich habe es aus veralteter Hardware gebaut. Als Betriebssystem dient ein Linuxsystem mit dem Programm tcpdump. Mit aktueller Hardware wäre es möglich, einen Sniffer zu bauen, der kleiner als ein übliches Handy ist und um Speicherplatz zu sparen, gezielt nur Passwörter, Kontodaten und andere sensible Informationen mitspeichert und per Funk an einen Angreifer sendet, wie es aktuell bereits bei Bankkartenlesegeräten vorkommt.

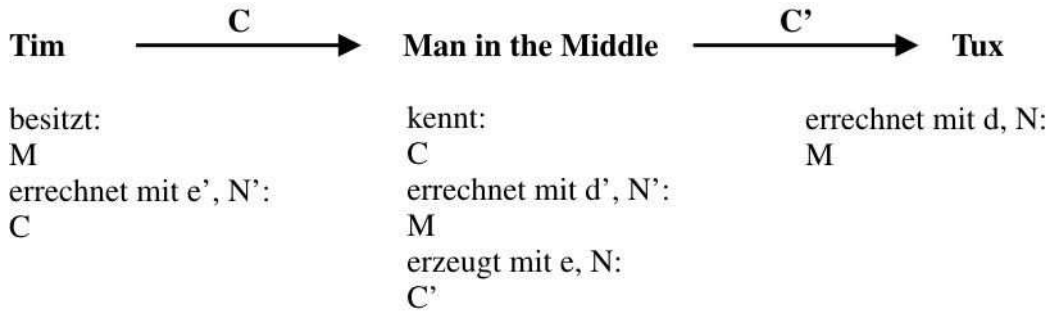
10. Anhang Man in the Middle

Ein Man-in-the-Middle (kurz: MitM) Angreifer (übersetzt: Mann in der Mitte) bezeichnet meistens einen **aktiven** Angreifer, der sich zwischen Tim und Tux geschaltet hat und in der Lage ist, ihre Pakete zu verändern. Diesem Angriff ist die beste asymmetrische Verschlüsselung nicht gewachsen, da es **kein Angriff auf die Verschlüsselung, sondern ein Angriff auf die Identität von Tim und**

Tux ist. Zum genauen Verständnis dieses Angriffs wird Kenntnis über RSA – mit den in der Facharbeit verwendeten Variablen – vorausgesetzt.
Der MitM gibt sich Tim gegenüber als Tux aus und Tux gegenüber als Tim aus. Damit ist es ihm möglich, Tim seinen eigenen Public-Key zu geben – den der MitM selbst erstellt hat – und somit das von Tim Verschlüsselte entschlüsseln kann.



Damit erhält er die verschlüsselten Daten von Tim, errechnet aus ihnen den Klartext und kann diese mit der von Tux geforderten Verschlüsselung auch an Tux schicken.



Es zeigt sich also, dass man mit einer asymmetrischen Verschlüsselung alleine keine sichere spontane Kommunikation einleiten kann.

11. Anhang Teilerfremd

> „*Zwei natürliche Zahlen a und b werden als **teilerfremd** (oder **relativ prim**) bezeichnet, wenn bei ihrer Primfaktorzerlegung kein gemeinsamer Faktor vorkommt.*“ [4]

4 Quelle: http://de.wikipedia.org/wiki/Teilerfremdheit

Es gilt:

$t \mid a$ und $t \mid b$

t heißt gemeinsamer Teiler von *a* und *b*. Wenn *t* seinen größtmöglichen Wert annimmt, wird *t* als größter gemeinsamer Teiler von *a* und *b* bezeichnet.

$ggT(a,b) = t$

Ist $ggT(a,b) = 1$ bedeutet das, dass *a* und *b* bei einer Primfaktorzerlegung keinen gemeinsamen Teiler haben, denn Eins ist keine Primzahl. Dies ist eine effektive Möglichkeit um *a* und *b* auf Teilerfremdeheit zu überprüfen:

$ggT(a,b)=1$

12. Anhang Erweiterter euklidischer Algorithmus

Definition Erweiterter euklidischer Algorithmus

> *„Zu jedem Paar ganzer Zahlen (a,b) gibt es ganze Zahlen x und y, sodass gilt:*
> *ggT(a,b) = x * a + y * b*
> *Das Verfahren zur Bestimmung dieser Zahlen x und y wird auch als das 'Erweiterte Euklidische Verfahren' bezeichnet.“* [5]

Es gibt viele Möglichkeiten zur Berechnung dieses Verfahrens, da es weitgehend optimiert und zur schnelleren maschinell-automatisierten Ausführung angepasst wurde. Für die von uns benötigte zahlentheoretische Grundlage reicht es allerdings zwei Sachen zu wissen:

1) Den erweiterten euklidischen Algorithmus erhält man am einfachsten, indem man den bereits berechneten euklidischen Algorithmus nochmals mit den Ergebnissen rückwärts rechnet und die Gleichung nur so weit vereinfacht, dass die beiden Zahlen *a,b* nicht mit *x,y,* oder miteinander verrechnet werden, sonst bekommt man die allgemeingültige Aussage 1=1.
2) Da ein Mensch niemals so schnell im Kopf etwas berechnen kann, wie es aktuelle Rechner können, braucht man nur das Prinzip des erweiterten euklidischen Verfahrens kennen, nicht wie es gegenwärtig implementiert wird.

5 Quelle: http://yimin.sinuslab.net/doc/mathematik_von_rsa.pdf; Seite 8

13. Anhang Lösung über den chinesischen Restsatz

„ed ≡ 1 mod φ(N)

Die multiplikative Eigenschaft der Eulerschen φ-Funktion führt dann zu

$$ed \equiv 1 \bmod p-1$$

Der modulo-Operator kann umgeschrieben werden. So existiert eine ganze Zahl k mit ed = k(p-1)+1. Dies wird in den zu beweisenden Ansatz eingefügt.

$$M^{ed} \equiv M^{k(p-1)+1} \equiv M*(M^{p-1})^k \bmod p$$

Nach dem kleinen Satz von Fermat gilt für alle zu p teilerfremden Zahlen M der Zusammenhang $M^{p-1} \equiv 1 \bmod p$. Mit Einsetzen und Umformen kommt man so zu

$$M*(M^{p-1})^k \equiv M*1^k \equiv M \bmod p$$

Somit wurde gezeigt, dass $M^{ed} \equiv M \bmod p$ gilt. Analog dieser Ableitung kommt man zu $M^{ed} \equiv M \bmod q$. Mithilfe eines Spezialfalles des chinesischen Restsatzes können nun beide Kongruenzen unter der Bedingung N=pq kombiniert werden.“ [6]

14. Anhang UTF-8 Zeichentabelle

Eine UTF-8 Zeichentabelle befindet sich beispielsweise unter folgendem Link:

http://www.utf8-zeichentabelle.de/

Diese Zeichentabelle ist allgemein gültig und sollte in jeden aktuellen Betriebssystem bereits implementiert sein.

15. Anhang Bit

Ein Bit ist in der Informatik im Kontext der Verschlüsselung eine Speichereinheit, welche nur die zwei Zustände 0 und 1 kennt. Acht Bits werden zu einem Byte gruppiert. Folglich lassen sich ein einem Byte $2^8 = 256$ Zustände speichern. In den Anfangszeiten des Computers war es nur mit einem Byte möglich, alle Buchstaben und Zeichen, die ein Schriftsatz benötigte, zu speichern. Heutzutage werden jedoch bis zu 4 Bytes benötigt, um ein Zeichen zu speichern, denn aufgrund der Globalisierung, werden in modernen Zeichensätzen – wie UTF-8 –

6 Quelle: http://de.wikipedia.org/wiki/RSA-Kryptosystem
a wurde in dem Original ersetzt durch *M* und *u* durch *k.*

Buchstaben, Sonderzeichen und Steuerungszeichen (wie Zeilenumbruch) aller Länder und Sprachen in einem Zeichensatz vereint (deren Anzahl aktuell weit über eine Millionen ist). Dennoch bleibt die Anzahl der Bits in der Kryptographie ein Maß für die Stärke des verwendeten Schlüssels. Normalerweise ist die Anzahl der Bits eine durch Acht teilbare Zahl, da die Informationen byteweise gespeichert werden. Ein Schlüssel der Stärke 2048 bietet somit 2^{2048} Kombinations-möglichkeiten aus denen er sich zusammensetzen lässt. Wenn man nun der Einfachheit her dem Verschlüsselungssystem einen einfachen 8bit Schriftsatz zugrunde legt, bestünde der Schlüssel somit aus $2048/8=256$ Zeichen aus der Menge [a-zA-Z0-9].

16. Anhang Heise News

Die angegebenen Zahlen stammen aus der Quelle:
http://www.heise.de/newsticker/meldung/6595
Ich möchte hier darauf hinweisen, dass dies keine statische URL ist, sondern sich die Zahl in der URL ändert und damit der Link nicht mehr aktuell ist. Eine Kopie der Webseite vom 09.11.2005 um 17:45 habe ich der Facharbeit beigelegt.