Spezialthema Kryptologie

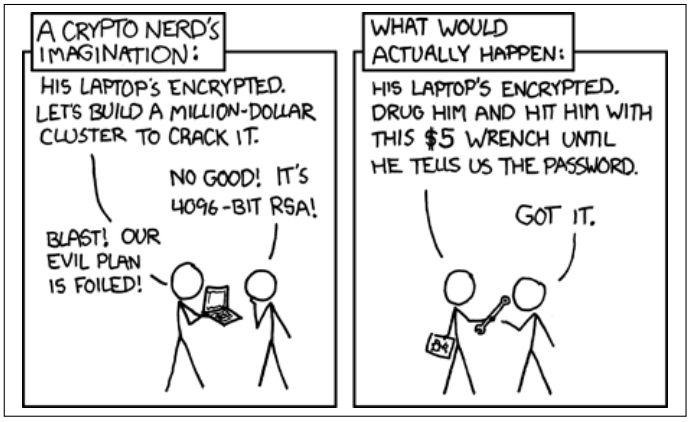
Lukas Prokop

19. November 2010

Dank an
Prof. Schmidhofer & Prof. Egger

"Codebrecher sind Alchemisten der Sprache, ein mythenumworbener Stamm,
der versucht, sinnvolle Worte aus bedeutungslosen Symbolreihen hervorzuzaubern"
(Simon Singh)

"If privacy is outlawed, only outlaws will have privacy"
(Phil Zimmermann)



"Security" `http://xkcd.com/538/`

Inhaltsverzeichnis

Kapitel 1

Kryptologie allgemein

1.1 Kryptologie – Ein Überblick

Kryptologie ist jene Wissenschaft, die sich mit dem Ver- wie auch Entschlüsseln von Nachrichten befasst. Seit jeher versuchen Menschen ihre Geheimnisse nur für sich selbst zu bewahren und entwickeln hierfür Algorithmen, um Nachrichten so zu verschleiern, dass niemand mitlesen kann. In diesem 99-seitigen Dokument möchte ich die Kryptologie ein bisschen beleuchten. Der erste Teil wird die Kryptographie allgemein beleuchten und ein paar Elemente erklären, um dann die alten kryptographischen Werkzeuge beschreiben zu können. Es handelt sich also um eine historische Betrachtung. Danach werden wir uns die kryptoanalytischen Techniken ansehen mit denen diese Systeme geknackt werden können. Und am Ende landen wir im großen Abschnitt der modernen Kryptographie. Im Zentrum dieses Dokuments steht das Kryptosystem RSA.

Die Abb. 1.1 stammt von mir aus dem März 08. Sie zeigt gut, welche Bereiche die Kryptologie umfasst. Zuerst teilt sie sich in die Kryptographie (Verschlüsseln) und Kryptoanalyse (Entschlüsseln) auf. Im Hintergrund der beiden Wissenschaften versteckt sich die Steganographie (Verschleiern) und ihr Gegenstück Steganalyse (Entschleiern). Die Kryptoanalyse besteht aus verschiedenen Methoden, die Kryptologen im Laufe der Jahre entwickelt haben. Die Häufigkeitsanalyse richtet sich gegen die monoalphabetische Substitution und Charles Babbage entschlüsselte den ersten Vigenére-Code. Kerckhoffs Prinzip betrifft sowohl die Kryptographie wie auch Kryptoanalyse und begründet die moderne Kryptologie. Die Kryptologie verliert sich nach dieser Zeit in der Größe der modernen Kryptologie. Beispiele für moderne kryptographische Methoden sind DES, AES, MD1-5, Diffie-Hellman und viele weitere. Ein Teil davon wurde bereits gebrochen und nur ein Bruchteil wird in diesem Dokument wird besprochen.

Der Aufbau des Dokuments ist grundlegend chronologisch aufgebaut, um einen besseren Überblick zu den Entwicklungen zu geben. Auch wenn dieses Dokument schon sehr viele Methoden und Algorithmen bespricht, so umfasst es nicht alles. Ich verzichte darauf

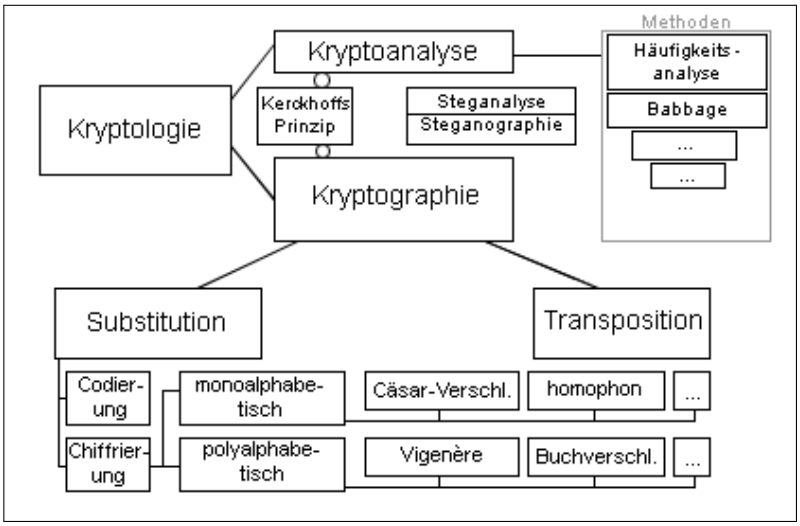


Abbildung 1.1: Einteilung der Kryptologie

bei bestimmten Kapiteln näher darauf einzugehen und wenn sich die Notiz (Stichwort "*Thema*") finden lässt, kann nach jenem *Thema* selbstständig gesucht, gegoogelt und geforscht werden.

1.2 Anwendung von Kryptosystemen

Anwendungen für Kryptologie findet man in unserem gewöhnlichen Alltag. Egal ob man sich auf einer Webseite einloggt oder in der Schule einen (für den Lehrer) unleserlichen Schummelzettel schreibt... die Kryptologie begleitet uns. Es sollte heute primäres Ziel der Kryptologen sein die Kryptologie zugänglich im Sinne von benutzerfreundlich zu machen.

Ursprünglich fand die Kryptographie vor allem im militären Bereich Motivation. Es war im Sinne der Herrscher an ihre Verbündete Nachrichten zu senden, die von einem Dritten nicht gelesen werden konnten. Die Kryptologie ist geprägt von einem ewigen Kampf zwischen Kryptographen und Kryptoanalytikern; jenen Personen, die eine Wissenschaft entwickelten, wie man diese Codes entwickeln und ebenso brechen kann.

1.3 Aufgabenstellung

In der Kryptologie geht es um ein Grundproblem: Wie kann ich dafür sorgen, dass eine (von mir verfasste) Nachricht nur von mir und meinem Freund gelesen werden kann? Wir werden entdecken, dass dies eine lösbare Aufgabe ist; jedoch immer nur im Rahmen von Bedingungen und Umständen. Ändern sich die Umstände, kann ein Kryptosystem

gegebenenfalls geknackt werden.

Nebenbei werden wir auch weitere Problemstellungen entdecken: Ich lege eine Nachricht ab, wobei ich nicht mehr die ursprüngliche Nachricht auslesen können möchte. Später möchte ich nur mehr fähig sein eine Nachricht zu erstellen und überprüfen ob sie mit der ursprünglichen übereinstimmt. Ich möchte verifizieren, ob mein Gegenüber wirklich mein Kollege ist.

Wir werden sehen, dass alle Problematiken recht ähnlich sind wir können die Ideen für die eine Problemstellung auf die andere anwenden.

1.4 Abkürzungen zur Beschreibung von Kryptosystemen

Bevor wir etwas tiefer ins Thema einsteigen, müssen wir noch ein paar Begriffe kennen lernen, die wir im folgenden Kapitel verwenden werden.

Kryptologie Der Begriff stammt von den griechischen Wörtern "krypto" für *geheim*, "logos" für *Wort oder Sinn* und Kryptographie leitet sich von "graphie" für *schreiben* ab.

Chiffre *chiffre* ist französisch und steht für Ziffer. Aufgrund der Beiträge der Franzosen zur Kryptologie wurde dieser Begriff Teil der kryptologischen Terminologie.

Algorithmus Algorithmen sind schrittweise Anleitungen zur Lösung eines Problems. Um einen Text zu verändern, implementieren wir Algorithmen, um einem Rechner Anweisungen zu geben, wie er das Problem zu beheben (bzw. den Text zu modifizieren) hat. Wir werden entdecken, dass es Algorithmen gibt, die das Problem nicht in endlicher Zeit lösen. Jene Algorithmen bezeichnen wir als "ineffizient".

Kodierung Siehe Abschnitt 1.5 auf Seite 9

Involution Der Algorithmus zur Verschlüsselung ist zugleich der Algorithmus zur Entschlüsselung

Monoalphabetik Wir verwenden bei der Kodierung und Dekodierung nur ein Alphabet

Polyalphabetik Wir verwenden bei der Kodierung und Dekodierung zwei bzw. mehrere Alphabete

Substitution von lat. substituere = ersetzen. Ein Zeichen wird durch ein anderes ersetzt.

Translation von lat. transponere = versetzen. Ein Zeichen nach einem bestimmen Ablauf versetzt.

Präfixcode ein Code, der die Fano-Bedingung erfüll. Siehe Sektion "Binärcode" auf Seite 10

Polybios-Matrix Der Klartext wird beispielweise zeilenweise in eine Matrix eingetragen und dann spaltenweise gelesen.

Ephemeralschlüssel Schlüssel, der nur einmalig verwendet wird

security by obscurity Diese Technik basiert auf der Annahme, dass die Vielfalt an Verschlüsselungsalgorithmen selbst schon für ausreichend Sicherheit sorgt. Wenn ich die Zeichenkette MIEHEG nenne, so hat der Kryptologe zahlreiche Möglichkeiten wie diese Kette zu entschlüsseln sein könnte. Security by obscurity wird von Kirckhoffs verboten, da der Algorithmus auch veröffentlichbar sein sollte

Nebenbei definieren wir noch Variablen für die mathematische Beschreibung. Je nach Kontext werden sie auch klein geschrieben. Die Buchstaben können sowohl für ihre Menge wie auch ein Element ihrer Menge stehen.

A, B zwei (nicht näher definierten) Mengen

I die Identität des ersten Elements eines Alphabets

E die Identität des letzten Elements eines Alphabets

P öffentlicher Schlüssel, public key

S geheimer Schlüssel, private key

M Klartext, Botschaft (bzw. ein Element davon)

C Code, verschlüsselter Text (bzw. ein Element davon)

q, p, n Primzahlen und natürliche Zahlen für die Verschlüsselungen

Mit diesen Variablen können wir ein Kryptosystem beschreiben. Im folgenden Beispiel wird der private Schlüssel auf den Klartext angewandt. Das Ergebnis ist der Code.

$$C = P(M)$$

1.5 Kodierungen

Das wichtigste Kriterium für ein funktionierendes Kryptosystem ist die eindeutige Zuordnung. Ein Element a der Menge A wird genau einem Element b der Menge B zugeordnet.

Dies ist notwendig, damit wir eine Nachricht im Alphabet A jederzeit ins Alphabet B translatieren können, ohne Daten zu verlieren. Das selbe sollte auch rückwärts funktionieren.

Das entspricht der Definition einer mathematischen Funktion. Als Beispiel möchte ich die Winkelfunktion Kosinus heranziehen. Ein Winkel $\alpha = 60°$ kann genau einem Cosinus-Wert $cos(60°) = 0.5$ zugeordnet werden. Allerdings ist die Äquivalenzumformung ("Rücktranslation") *arccos* mehrdeutig. Wenden wir sie auf 0.5 an, erhalten wir *2* Werte (60°, 300°)

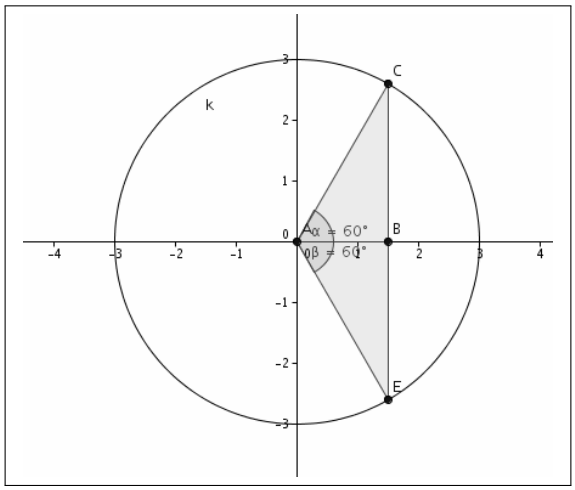


Abbildung 1.2: Mehrdeutigkeit der Winkelfunktionen *cos* und *arccos* im Einheitskreis

Wir erkennen also die Verletzung der Definition der "eindeutigen Zuordnung". Allerdings bezeichnen wir sie trotzdem so, weil sie zumindest in eine Richtung eindeutig ist und somit eine "Einwegfunktion" ist. Einwegfunktionen werden erst in der Sektion Hashes eine Rolle spielen. Wir bleiben vorerst bei Eindeutigkeit und damit bei der Entzifferbarkeit von Codes.

Eindeutigkeit ist auch das Kriterium für einen anderen Begriff. Bei der "Kodierung" wird ein Text im Alphabet A in einen Text des Alphabets B übersetzt. Dabei ist die Translation von A nach B – wie auch umgekehrt – eindeutig. Deshalb können wird jede valide Verschlüsselung auch Kodierung nennen.

Ich möchte ein paar Kodierungen vorstellen. Man könnte natürlich einfach das griechische Alphabet statt dem lateinischen verwenden oder die ASCII-Zeichen verwenden. Um ein bisschen Geffuhl für die Verwendung von Kodierungen zu geben, möchte ich jedoch drei Kodierungen vorstellen, die heute in bestimmten Bereichen nach wie vor eingesetzt werden. Mit ihnen kann eine einfache monoalphabetische Substitution durchgeführt werden.

1.5.1 Morsecode

Der Morsecode basiert wie der Binärcode auf einem Minimalalphabet mit 2 Zeichen. Allerdings wird auch ein Trennzeichen zwischen den Zeichen benötigt. Deshalb besteht es eigentlich aus 3 Zeichen.

A	.-		N	-.
B	-...		O	—
C	-.-.		P	.–.
D	-..		Q	–.-
E	.		R	.-.
F	..-.		S	...
G	–.		T	-
H	....		U	..-
I	..		V	...-
J	.—		W	.–
K	-.-		X	-..-
L	.-..		Y	-.–
M	–		Z	–..

Da T und E die häufigst verwendeten Zeichen sind, werden ihnen die kürzesten Zeichenfolgen zugewiesen. Am bekanntesten ist der Funkspruch `... --- ...` für SOS, welcher für "Save our souls" steht. Morsecode wird noch aktuell in der Schifffahrt eingesetzt und in Gefahrensituation wird jener Funkspruch mit Lichtzeichen abgegeben.

Ich habe gerade in einem alten Ordner einen Text gefunden. Mein Großvater hat den Morsecode fließend beherrscht und wollte ihn mir beibringen. Auf dem Zettel steht:

```
.-.. ..- -.- .- ...
.--. .-. --- -.- --- .--.
```

1.5.2 Binärcode

Der Binärcode besteht aus dem Minimalcode der Menge {0,1}. Der Binärcode ist elementarer Baustein der Kommunikation, da binäre Zeichen (bzw. boolsche Werte) gut als elektrische Signale versandt werden können. Die Zahlen Null und Eins werden auch oft als Wahr (True) wie auch Falsch (False) interpretiert. Bei Schaltern spricht man von An (On) und Aus (Off). Wir könnten nun die ganze Schaltalgebra (boolsche Algebra) besprechen, aber das würde den Rahmen des Dokuments sprengen. Ich möchte eine spezielle Bedingung besprechen, um das Interesse an der Kodierungstheorie zu erwecken: die Fano-Bedingung.

Die Frage wieso der Morsecode ein drittes Zeichen benötigt, ist auf die Nichterfüllung der Fano-Bedingung zurückzuführen. Eine Zeichenkette `.--.` ist mehrdeutig. Sie könnte

sowohl stellvertretend für "an" wie auch "we" oder einfach nur "p" stehen. Die Forderung der Fano-Bedingung: Ein Zeichen x darf nicht in zwei Zeichen y und z zerlegt werden können, wobei weder $y, z \in A$ sein darf, noch y und z Teil eines Elements aus A.

Während der Morsecode von Haus aus diese Bedingung nicht erfüllen kann, ist der Binärcode anpassbar. Die folgenden Mengen erfüllen die Fano-Bedingungen:

$$A_1 = \{0, 1\}$$

$$A_2 = \{0, 100, 101\}$$

$$A_3 = \{1, 10, 100\}$$

$$A_4 = \{1, 101, 1001, 10001\}$$

```
import random
print ''.join([str(random.choice((1, 10, 100))) for _ in xrange(3)])
```

Das angeführte python-Skript berechnet uns in Sekundenschnelle eine zufällige Zeichenkette der Menge A_3 (bei mir hat es 11100 ergeben). Wir betrachten das erste Zeichen (1). Jenes Zeichen sagt eigentlich gar nichts aus, da es Teil jedes Elements aus A_3 sein könnte. Das zweite Element (1) kann gar nicht Teil von einem anderen Zeichen sein; das erste Zeichen steht alleine und das zweite Zeichen leitet ein neues ein. Wir können also "1 1" notieren. Das nächste Element ist ebenfalls eine Eins. Das zweite Element musste also wiederum für sich selbst stehen ("1 1 1"). Das vierte Zeichen könnte sowohl Teil von Element 2 wie auch Element 3 sein. Da noch eine Null folgt, muss es sich um Element 3 handeln. Wir konnten die Nachricht "11100" eindeutig in "1 1 100" zerlegen. Eine eindeutige Zuordnung (Kodierung) konnte erfolgen. Wir können noch zahlreiche andere Zeichenketten generieren lassen: Es wird sich zeigen, dass A_3 ein sogenannter Präfixcode ist.

Der Huffman-Code versucht sowohl die Fano-Bedingung zu erfüllen wie auch eine Optimierung aufgrund der Häufigkeit der Zeichen durchzuführen.

Wie umgeht man die Fano-Bedingung? Nachdem durch die Fano-Bedingung einige Möglichkeiten weg fallen, wird bei Datentypen die Speicherverwaltung anders organisiert. Die wichtigste Lösung zu dem Problem ist die einheitliche Länge. Wenn alle Elemente der Liste gleich lang sind, sind alle Kombinationen zu nutzen, allerdings fallen jene Kombinationen weg, die weniger Zeichen beinhalten. Eine Zufallszeichenkette der folgenden Liste kann immer eindeutig zerlegt werden, wobei keine Einschränkungen bezüglich des Präfixes erfüllt werden müssen:

$$M = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

1.6 Transposition

Die Transposition ist das Gegenstück zur Substitution. Die Buchstaben werden nicht ausgetauscht, sondern anders positioniert. Die einfachste Form ergibt sich, wenn man die Buchstaben abwechselnd in zwei Reihen schreibt.

T	a	s	o	i	i	n	c	i	f	e
r	n	p	s	t	o	s	h	f	r	

Danach könnten wir die zwei Reihen einfach aneinander reihen: "Tasoiincifernpstoshfr". Um die Zeile lesen zu können, definieren wir n als die Anzahl aller Zeichen dividiert durch die Anzahl der Reihen ($n = \frac{22}{2} = 11$). Um die Nachricht zu lesen müssen wir zuerst Buchstabe $0 \cdot n$ lesen. Danach $1 \cdot n$, $0 \cdot n + 1$ und $1 \cdot n + 1$, $0 \cdot n + 2$ und $1 \cdot n + 2$ usw.

Eine andere Variante wäre es das Wort nach zB 4 Zeichen umzubrechen. Daraus ergeben sich 4 Spalten. Diese 4 Spalten können wir anders ordnen. Und dann die Zeichen wieder reihenweise zusammenhängen. Diese Variante habe ich der ADFGVX-Verschlüsselung (Kapitel 2.4) entnommen.

Die Praxis zeigt, dass Transposition bei der Anzahl der Möglichkeiten wesentlich höhere Werte aufweisen kann, jedoch besteht das Grundproblem, dass ein Schlüssel schwer formulierbar ist. Während bei der (monoalphabetischen) Substitution ein Alphabet der Schlüssel ist, muss bei der Transposition eine ganze Beschreibung des Vorgangs mitgeliefert werden (zB alpabetische Ordnung der Spalten). Man könnte natürlich die Beschreibung auf nur ein Wort reduzieren (zB Schlüsselwort ist "Krypto" und Teil des Algorithmus ist die alphabetische Sortierung). Wenn allerdings dieses Wort in die Hände des Gegners fällt, kann er die Nachricht ganz einfach dechiffrieren. Die Sicherheit darf nicht auf der Geheimhaltung des Algorithmus' basieren (das wäre security by obscurity) (siehe Kerckhoffs auf Seite 35)

Alte Kryptosysteme können mithilfe der Begriffe Substitution und Transposition klassifiziert werden. Moderne Algorithmen verwenden allerdings alle möglichen Kombinationen und Varianten von Substitution und Transposition zugleich. Eine Klassifizierung oder eine nähere Betrachtung mithilfe dieser Begriffe scheint nicht mehr sinnvoll zu sein.

1.7 Steganographie

Die Steganographie verfolgt das Ziel die Existenz einer Nachricht zu verschleiern. In der Kryptographie sollte man eine Nachricht in die Hände bekommen können und trotzdem nichts mit dem Inhalt anfangen können. Das Ziel der Steganographie wurde bereits verfehlt, wenn der Gegner die Nachricht in die bekommt. In Agentenfilmen wird gerne ein geheimer Agent alleine mit der Ware losgeschickt, während ein Cobrateam eine Fälschung auf einem anderen Weg zum Zielpunkt transportiert. Auch dies ist eine Form

der Steganographie.

Den Sklaven der Antike wurde der Kopf geschoren. Die Nachricht wurde dann auf ihre Kopfhaut tätowiert. Als die Haare nachgewachsen waren, wurden die Sklaven versandt. Der Empfänger musste sie nur wieder die Haare rasieren lassen. Dieses Verfahren gilt als das älteste der Steganographie und stammt aus der Zeit des Cäsarcodes.

Weitere Verfahren:

- **Wahrnehmungsschwelle:** Viele Techniken nutzen die Tatsache, dass unsere Wahrnehmung nur sehr eingeschränkt ist. Zu kleine Objekte sehen wir nicht. Zu leise oder hohe und tiefe Töne nehmen wir nicht wahr. Genauso nehmen wir nur bestimmte Düfte wahr
- **Doppelter Boden:** In Taschen baut man doppelte Böden ein, damit man Waren unbemerkt durch Kontrollen schmuggeln kann
- **unsichtbare Tinte:** Der Autor schreibt die Nachricht mit unsichtbarer Tinte (zB Zitronensaft) und lässt dann sein Blatt trocken. Das scheinbar weiße Blatt Papier wird überliefert und der Empfänger legt es am Heizkörper. Der Zitronensaft färbt sich bräunlich und die Nachricht wird lesbar
- **Digitale Daten:** Ganz allgemein lassen sich in digitalen Daten gut geheime Informationen verstecken. Syntaktisch falsch Codesequenzen (die nicht der Spezifikation entsprechen) werden oft nicht interpretiert und die Information wird dadurch nicht sichtbar.

Ich möchte eine besondere – selten verwendete – Form der Steganographie ansprechen. Steganographie und digitale Texte sind so gut wie nie vertreten, aber dieser Klartext lässt jeden Kryptographen verzweifeln. Bei *beatnik* handelt es sich um eine esoterische Programmiersprache. Jene Programmiersprachen haben es nicht zum Ziel effizientes Programmieren zu ermöglichen, sondern sollen nur die Basiskonzepten der Programmierung demonstrieren. In dem Fall handelt es sich um eine turingvollständige Programmiersprache. Jedes Problem welches man mit höheren Programmiersprachen lösen kann (Stichwort "Alan Turings Komplexitätstheorie"), kann man auch in Beatnik lösen (auch wenn nicht so effizient).

Und wer hätte gedacht, dass dieser Klartext die Ausgabe "Verschluesslung" produziert?

```
Grüss Sie Gott, Kommission

Unser Professor Mat Schmiddy unnd Professor Teres Magizter Egger
grüssen schüler. Sind alle dort? Einige Maturanten sind schon
neugierig. Wenn meine Mathura bessa viiiiiel meehr rockt, dann
bin ik teng. Fredd meint ddass uund Wwindows sucss. Linux succs
inkiner Wise. Ikk fraage mich mnches. Mit Saxophone spieldt.
```

```
hhh ppp
hhh add bbbinar bbe ppp
hhh add ddddddd bbe ppp 7
hhh add un kk ppp
hhh add tri bbe ppp
hhh add ddde bbe ppp
hhh add hhhh bbe ppp
hhh ppp
hhh add ggggggg bbe ppp 13
hhh add ddddddd bbe ppp 14
hhh ppp
hhh add eeeeeee bbe ppp
hhh add vvvv bbe ppp
hhh add ccc bbe ppp
hhh add no bbe ppp

... aaand thanksalot
to all the geeks around the world
```

Beatnik hat die Funktionsweise Wörter wie im Gemeinschaftsspiel Scrabble zu bewerten. Je nach Häufigkeit des Buchstaben wird ihm eine Punkteanzahl verliehen. Die häufig auftretenden Umlaute (vor allem im Englischen) a, e, i, o und u bekommen bloß einen Punkt. h ist seltener und bekommt 4 Punkte und z und q sind ganz selten (10 Punkte). "hhh" bedeutet also $4+4+4=12$ und der Befehl 12 steht in Beatnik stellvertretend für "duplicate". Der oberste Wert des Stacks wird dupliziert und noch auf den Stack gelegt.

Was ist ein Stack? Ein Stack ist wie ein Bücherstapel. Man kann zwar das oberste Buch runternehmen oder ein neues drauflegen (bei unserem Beispiel eine Zahl), aber man kann kein Buch in der Mitte herausreißen. Dann würde der Stapel zusammenfallen. Stacks sind ein Grundbestandteil von Computern. In Beatnik haben alle 26 Buchstaben einen Wert zugewiesen bekommen und jede Bewertung ist einem Stack- bzw. Beatnikbefehl zugewiesen. Unbekannte Buchstaben (wie ü in *Grüss Gott*) werden ignoriert.

Mein Klar- bzw. Quelltext besteht aus 2 Teilen. Im ersten Teil habe ich versucht einen möglichst realistischen Text zu verfassen. Ich zweiten Teil habe ich demonstriert, wie man den Quelltext noch undeutbarer macht. "thanksalot" steht für die Zahl 17 und terminiert das Programm. Ich habe bereits erwähnt, dass "hhh" für "duplicate" steht. Dass Zeile 7 und 14 den selben Buchstaben (s) ausgibt, mag vielleicht erkennbar sein, aber dass Zeile 13 die selbe Aktion ausführt, erscheint nicht mehr so logisch, wenn man die Funktionsweise von Beatnik nicht kennt. Ich hätte auch den gesamten Quelltext nur mit "a" schreiben können. Ich wollte damit demonstrieren, dass der Zweck des Quelltexts damit unerkennbar wird und somit steganographisch wirkt.

Apropos "Funktionsweise kennen"... die meisten steganographischen Algorithmen basie-

Abbildung 1.3: Scrabble mit Buchstabenwertigkeiten

ren auf dem Konzept der *security by obscurity*. Das ist der Grund, wieso Steganographie in der digitalen Welt nicht stark vertreten ist. Realistische Beispiele gibt es dennoch. Wasserzeichen kennzeichnen das Copyright von Photographen. Oder wir können die Farben eines Bildes Zeile für Zeile auslesen und jeder Farbe einem Index zuordnen (zB ergibt sich der Index aus der Summe der RGB-Farbwerte). Je nachdem ob es sich um eine gerade oder ungerade Zahl handelt, merken wir uns eine 0 bzw. 1. Auf diese Weise können wir eine Binärinformation speichern. Wenn selbst eine Nachricht schreiben wollen, dann ändern wir die Farbwerte um einen Index. Wenn wir beispielweise den Rotwert um einen Wert erhöhen, ist das für das Auge nicht mehr erkennbar, aber die Information wird trotzdem gespeichert. Deshalb hat man auch immer Angst und setzt Kryptologen ein, wenn Terroristen Videobotschaften von ihren Anführern senden.

Die Wissenschaft, die sich mit dem Analysieren von steganographischen Mitteln befasst, nennt sich *Steganalyse*.

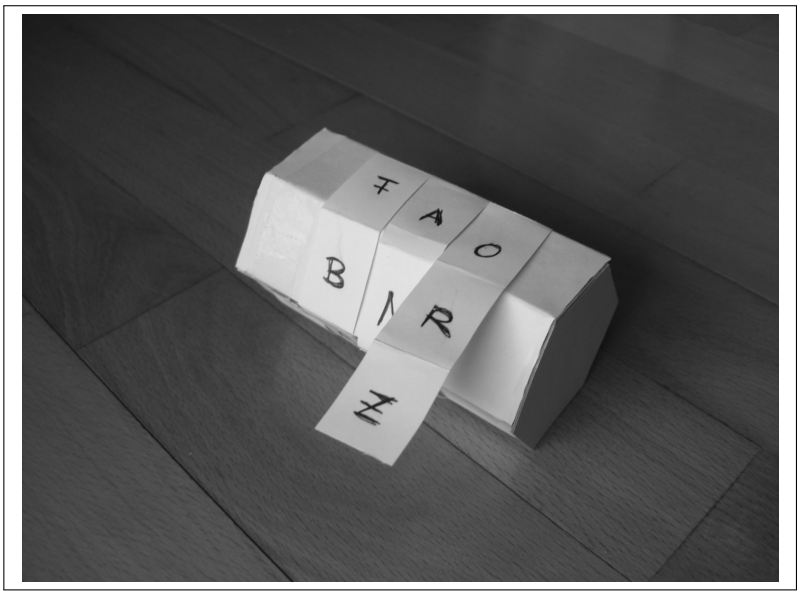
Übrigens... ist in der Bibel auch etwas versteckt worden? (Stichwort "Bibelcode")

Kapitel 2

Kryptographie

2.1 Skýtala von Sparta

Die Skýtala hat ihren Namen von der Stadt Sparta und wurde von Plutarch überliefert. Sie wurde 2500 v.Chr. von der spartanischen Regierung eingesetzt. Es war der erste Prototyp einer Transpositionschiffre; einer Verschlüsselung die auf der Permutation der Buchstaben eines Klartexts entspricht. Da die Zuordnung einem genau definierten System folgt, ist die Entschlüsselung auch wieder möglich.



Die Skýtala von Sparta ist ein Prisma (mit sechseckiger oder runder Grundfläche), um welches eine Papierrolle spiralförmig aufgewickelt wird. Die zu verschlüsselnde Nachricht wird Zeile für Zeile niedergeschrieben. Am Ende wird die Papierrolle von der Skýtala

genommen und verschlüsselungstechnisch bedeutet es, dass die Nachricht aus jedem $x$ten Buchstaben besteht (x ergibt sich aus dem Umfang der Skýtala). Gelangen wir zum Ende der Zeile, lesen wir jeden $(x + 1)$ten Buchstaben. Der Umfang der Skyt́ala wurde von den Spartanern als Skyt́ale bezeichnet.

Der "Schlüssel" ist die Skyt́ale. Man muss im Besitz eines identen Nachbau der ursprünglichen Skytala sein, um die Nachricht dekodieren zu können. Der Empfänger braucht die Papierrolle nur spiralförmig auf die Skytala zu spannen, um die Nachricht wieder zeilenweise lesen zu können.

> Eyonpnctri

2.2 Cäsarcode

Der Cäsarcode ist der bekannteste Algorithmus als Beispiel für monoalphabetische Verschiebechiffren. Gegenüber der Skytala werden die Buchstaben nicht transpositioniert, sondern für stellvertretende Buchstaben ausgetauscht. Wir sprechen von einer "Substitution".

Den Namen bekommt der Algorithmus vom römischen Feldherr und Herrscher C. Julius Cäsar (100 bis 44 v.Chr.), der ihn als einer der ersten Personen zur Nachrichtenübertragung genutzt haben soll. Zeit seines Lebens konnte kein erfolgreicher Angriff auf den Algorithmus erfolgen und es ist bekannt, dass er den Schlüssel 3 (eine Verschiebung um 3 Stellen) verwendet haben soll. Sueton schrieb:

> Exstant es [epistolae] ad Ciceronem, item ad familiares de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet.

Wir erhalten den Cäsarcode indem wir das Alphabet um x Stellen verschieben. Die Anzahl der Stellen bildet dabei den Schlüssel und ist eine Zahl zwischen 1 und 26 (0 verschiebt das Alphabet nicht und höhere Zahlen sind nicht sinnvoll, da der Modulo gebildet wird).

Alphabet	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
Cäsarcode	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

Tabelle 2.1: Eine Tabelle für den Cäsarcode mit dem Schlüssel 3

Möchten wir also den Text LUKAS verschlüsseln, ergibt das den Code OXNDV. Wir können bereits sehen: Ohne Tabelle ist das Rekonstruieren des Klartexts schon schwer.

Mit ein bisschen Übung und einem nicht allzu großem Schlüssel lässt sich die Nachricht ohne Papier entschlüsseln. Auf einem Computer werden solche Rotationen natürlich in Millisekunden errechnet. Im Kapitel Kryptoanalyse lernen wir einige Angriffe kennen. Der Cäsarcode ist ein klassisches System für eine unsichere Verschlüsselung. Seit dem 7. Jahrhundert n.Chr. gilt es als geknackt. Der Trick liegt darin, dass die Buchstaben auch die Eigenschaft Häufigkeit besitzen.

Mathematisch gesehen haben wir hier eine Addition von Buchstabenwerten. Die Informatik veranschaulicht sehr gut wie Zeichen in Zahlen kodiert werden. Wird die Zahl für A (zB 1) mit der Zahl 3 addiert, erhalten wir die Zahl 4. 4 wäre dann wieder zu einem D zu übersetzen. Es ergibt sich die Formel:

$$C = M + P$$

Jedoch müssen wir bedenken, dass das Alphabet nach dem Z aufhört und wieder von vorne anfangen sollte. $Z + 1$ sollte A sein. Damit wir dies erreichen, können wir den Rest einer Division auf die Größe des Alphabets anwenden; den Modulo. I bezeichnet den Index; den ersten Buchstaben des Geheimalphabets. E bezeichnet das Ende; den letzten Buchstaben des Alphabets.

$$C = M + P \bmod E$$

Wie bereits erwähnt, handelt es sich dabei um den Rest einer Division. 25 mod 26 ist 25, aber 27 mod 26 ist 1. Des Weiteren muss man bedenken, dass nicht jedes Alphabet mit 1 als erstem Wert beginnt. Bemerke, dass der Ausdruck $(E - I + 1)$ nur die Länge des Alphabets berechnet und I nur dazu vorkommt, um den Anfang des Alphabets zuvor auf 0 zu verschieben und später wieder auf den Ursprungswert zurückzubringen.

$$C = I + ((M + P - I) \bmod (E - I + 1))$$

Das schaut schon etwas komplexer aus. Zum Entschlüsseln müssen wir die Formel umkehren. Auch hier sollte bedacht werden: Das Alphabet hört nach $(E - I + 1)$ Zeichen auf.

$$M = I + (((C - P) - I) \bmod (E - I + 1))$$

Schaut schon etwas komplexer aus, aber wenn man alle mögliche Szenarien (Schlüssel größer 26, $C - P$ kleiner Null) durchgeht, ist die Formel herleitbar. Auf jeden Fall hätte Cäsar nicht an eine solche Komplexität gedacht.

Für Computer ist wichtig, dass ASCII-Großbuchstaben mit 65 beginnen. Ich verwende bei der folgenden Formel die Funktion ord(), welche den ASCII-Wert eines Buchstabens zurückgibt und chr() den Buchstaben eines ASCII-Werts. Dies entspricht den Funktionsnamen in höheren Programmiersprachen; insbesondere python.

$$M = chr(ord(I) + ((ord(C) - P - ord(I)) \% (ord(E) - ord(I) + 1)))$$

Für Programmierer: Bedenke, dass nicht nur Großbuchstaben verschlüsselt werden können sollen. Auch Kleinbuchstaben und Sonderzeichen sollten das Verschlüsselungsprogramm nicht zum Absturz bringen. Befasse dich mit ASCII (American Standard Code for Information Interchange)!

Der Modulo wird in Programmiersprachen meist mit einem Prozentzeichen dargestellt.

ROT13

Eine Sonderform der Cäsarverschlüsselung ist ROT13. ROT13 steht für "rotation 13" und nicht etwa die Rotverschiebung aus der Astronomie. Eine Verschiebung um 13 Stellen hat eine besondere Eigenschaft: Verschlüsselungs- und Entschlüsselungsalgorithmus sind ident (*Involution*), da 13 die Hälfte der Länge des lateinischen Alphabets ist. In Internetforen wird gerne mit ROT13 verschlüsselt, um einen Inhalt auf den ersten Blick unkenntlich zu machen. Browser-Plugins[1] übernehmen dann die Algorithmenarbeit. Das UNIX-Kommando tr bietet ein Werkzeug zum Verwechseln der Buchstaben an. Das folgende Beispiel konfiguriert ROT13:

```
user@linux ~ % tr A-Za-z0-9 N-ZA-Mn-za-m0-9
ROT13
EBG13
FooBar
SbbOne
^D
```

Alphabet	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
Cäsarcode	N O P Q R S T U V W X Y Z A B C D E F G H I J K L M

Tabelle 2.2: Eine Tabelle für ROT13

Zum Mitdenken!

ROT13 wurde bereits geknackt. Verschlüsseln Sie Klartexte deshalb bitte doppelt mit ROT13... zu ihrer eigenen Sicherheit.

[1] Leet Key ist ein Addon für den Webbrowser Firefox
https://addons.mozilla.org/en-US/firefox/addon/770

2.3 Vigenère

7. Jahrhundert Cäsarcode geknackt. Die Kryptoanalytiker haben gewonnen. Zur Zeit der Renaissance empfiehlt Leon Battista Alberti ein Kryptosystem, welches zwischen mehreren Alphabeten wechselt. Ein solches System wäre dadurch sicher, dass die Buchstaben nicht dem gleichen Schlüssel unterliegen und dadurch die Eigenschaft Häufigkeit verlieren. Es ergeben sich mehrere Alphabete ("polyalphabetische Verschlüsselung"). Eine Realisierung der Idee konnte er zu Lebzeiten nicht mehr finden.

Die Spur der polyalphabetischen Idee lässt sich von Alberti über Johannes Trithemius und Giovanni Porta bis zu Blaise de Vigenère nachvollziehen. Vigenère kam auf die Idee ein Schlüsselwort so oft zu verlängern bis es die Länge des Klartexts erreicht hat. Jeder Buchstabe des Klartexts wird dann mit dem Buchstaben des Schlüssels addiert. Als Tabelle kann man das Vigenère-Quadrat verwenden.

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
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	A
C	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
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
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	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	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

Tabelle 2.3: Das Vigenère-Quadrat (auch bekannt als Tabula recta)

Die Idee der polyalphabetischen Substitution löste tatsächlich das Problem der Auftrittswahrscheinlichkeit, wenn man den gesamten Code betrachtet. Jedoch ist es nur eine Sammlung von monoalphabetischen Substitutionen. Wir können also (wenn wir die Länge des Schlüssels kennen) den Geheimtext aufsplitten und wir erhalten genau (Länge des Schlüssels) monoalphabetische Substitutionen. Die Länge des Schlüssels drehen wir so lange hinauf, bis ein Alphabet die gewohnte Häufigkeitsverteilung aufweist. Die Länge des Schlüssels ist dann ein Vielfaches der Position des Alphabets. Im Bereich Kryptoanalyse werden wir die Kryptologen Kasiski und Friedman kennen lernen, die den Prozess der Schlüsselfindung erleichtert haben. Jedoch bemerken wir bereits zwei Schwächen bei den Angriffen:

- Es ist nicht ausgeschlossen, aber eine Implementierung für Computer gestaltet sich

schwierig, weil eine Bewertung der Situation (Welches Alphabet ist am nähesten der normalen Häufigkeitsverteilung) notwendig ist. Mit einer KI (Künstlichen Intelligenz bzw. Artificial Intelligence) sollte das Problem lösbar sein (verwende die empirische Standardabweichung).

- Bei kleinen Schlüsseln erhalten wir nur wenige Alphabete und viele Buchstaben. Die Häufigkeit dürfte nicht allzu verfälscht sein. Wenn wir jedoch einen langen Schlüssel und kurzen Text verwenden, dann gibt es für (beinahe) jeden Buchstaben ein eigenes Alphabet. Wir erkennen keine Perioden mehr und die Angriffe werden nicht erfolgreich sein.

Charles Babbage gilt als die erste Person, die einen Vigenère-Code entschlüsselt hat. Babbage ist einer der extrovertierten Wissenschaftler des 19. Jahrhunderts und machte verschiedene Entdeckungen und Erfindungen. Er berechnete, dass die Berechnung der Weglänge für jeden Brief aufwendiger ist als wenn man einen Einheitspreis fixiert. Er entwickelte auch eine "Differenzmaschine No. 1" (1823), die Gleichungen lösen konnte. Die Maschine legte theoretische Grundlagen für die Verarbeitung von Daten und Rechnerarchitektur. Ada Lovelace wurde seine Mitarbeiterin und auf seiner Basis entwickelte sie die erste Programmiersprache der Welt ("Ada"). Allerdings brach die britische Regierung die Unterstützung von Babbages Erfindungen ab und die digitale Revolution verschob sich nach hinten.

Charles Babbage wurde als Kryptoanalytiker in London bekannt und Menschen überhäuften ihn mit verschlüsselten Nachrichten. Bei der Vigenère-Verschlüsselung setzte der Auftraggeber (der sich nur überzeugen wollte, dass die Verschlüsselung sicher ist) zu früh ein Lächeln auf. Er nutzte seine statistischen Werkzeuge (Babbage war spezialisiert auf Statistik), um die Schlüssellänge zu filtern. Die Methoden können im Kapitel Kryptoanalyse nachgeschlagen werden. Hat man einmal die Schlüssellänge, ist es ein leichtes eine Zerlegung durchzuführen und die Häufigkeitsanalyse anzuwenden.

In einem Briefwechsel mit einem großen Dichter entschlüsselte er ein Gedicht, welches mit dieser Strophe endet[2]:

> Füllt den Krug, füllt den Becher:
> Auf ein Gelage vor dem Morgen:
> Jeden Augenblick stirbt ein Mensch,
> jeden Augenblick wird ein Mensch geboren.

Nicht dass Babbage nur das Schlüsselwort (den Namen der Frau Emily des Dichters) entdeckte, sondern als Statistiker informierte Babbage ihn über die falsche Annahme, dass die Bevölkerung stagniere:

> "Jeden Augenblick stirbt ein Mensch,
> jeden Augenblick wird $1\frac{1}{16}$ Mensch geboren."

[2] frei zitiert nach [6, S. 103]

> Die tatsächliche Zahl ist so lang, daß ich sie nicht
> auf eine Zeile schreiben kann, doch ich glaube, die Zahl $1\frac{1}{16}$ ist für die Poesie hinreichend genau.
>
> Immer der Ihre, Charles Babbage

2.4 ADFGVX

Als die mono- und polyalphabetische Verschlüsselung geknackt wurden, war die Kryptographie auf ihrem historischen Tiefstand. Kein Erfindergeist konnte neue Chiffren hervorbringen, die gegen die Kryptoanalyse eine Chance hatten. Eine Variation von schlechter Chiffre war die ADFGVX-Verschlüsselung, die von den Mittelmächten im 1. Weltkrieg eingesetzt wurde. Im Prinzip handelte es sich um eine zweischichtige Anwendung von Substition und Transposition. Der resultierende Code besteht nur aus den Zeichen A, D, F, G und X. Entwickelt wurde die Chiffre vom deutschen Offizier Fritz Nebel.

Als ersten Schritt erstellen wir eine Tabelle für die monoalphabetische Substitution.

	A	**D**	**F**	**G**	**V**	**X**
A	c	o	d	i	e	r
D	u	n	g	a	b	f
F	h	j	k	l	m	p
G	q	s	t	v	w	x
V	y	z	9	8	7	6
X	5	4	3	2	1	0

Tabelle 2.4: Tabelle für die ADFGVX-Verschlüsselung

Wie wir sehen können ist auf dem ($6{\cdot}6 = 36$)-Feld genug Platz für alle Kleinbuchstaben und Zahlen ($26 + 10 = 36$); jedoch nicht für deutsche Umlaute. Wie die Buchstaben j, p, v, x, y und q kommen auch Umlaute mit einer Häufigkeit kleiner 1% selten vor. Nachrichten lassen sich ebenso lesen, wenn Bchstben fehlen, Buchstaben (ß) lassen sich auch ersetzen (ss) oder um Umlaute auszulassen könnte man einen Text auch ins Englisch übersetzen. Da die Verschlüsselung im Krieg gegen England eingesetzt wurde, wurde auf diese Technik in diesem Fall verzichtet. Man muss bedenken, dass im 1. Weltkrieg keine Romane übertragen wurden und deshalb auch der Klartext nicht viele verschiedene Buchstaben enthielt.

In dieser Tabelle haben wir das Wort "codierung" zeilenweise eingetragen und die Tabelle mit den restlichen Buchstaben (abfhj...) aufgefüllt. Die Tabelle bildet den ersten von zwei Schlüsseln. Das stellvertretende Buchstabenpaar wird mit (Zeile, Spalte) angegeben.

In der unteren Zeile sehen wir das oben angesprochene *Buchstabenpaar*. Wir schlagen also den Buchstaben nach und schreiben das Buchstabenpaar (bestehend Zeilen- und Spaltenbezeichnung) darunter. Die Buchstabenpaare sind unser neuer Code.

p	a	i	n	v	i	n
FX	DG	AG	DD	GG	AG	DD

Tabelle 2.5: "painvin" ADFGVX-verschlüsselt

Die Länge des bisherigen Codes ist 14. Für die Transposition benötigen wir einen zweiten Schlüssel; ein Wort dessen Länge ein Teiler von 14 ist. Wir wählen das Wort "WLTKREG" (Länge 7).

W	L	T	K	R	E	G
F	X	D	G	A	G	D
D	G	G	A	G	D	D

Tabelle 2.6: Monoalphabetischer Code in einer Polybios-Matrix

E	G	K	L	R	T	W
G	D	G	X	A	D	F
D	D	A	G	G	G	D

Tabelle 2.7: Sortierung der obersten Elemente

Zur Transposition ordnen wir die Spalten alphabetisch, wobei wir uns an den Elementen der ersten Reihe orientieren (EGKLRTW). Der verschlüsselte Text lautet nun spaltenweise abgelesen (GDGGGAXGAGDGFD).

Da der Code nur aus 6 Zeichen besteht, wird die Verschlüsselung als klassisches Beispiel für senäre Kodierungen verwendet. Jedoch verwendete eine frühe Entwicklung der Verschlüsselung lediglich 5 Zeichen (ADFGX) und war somit ursprünglich quinär.

Bezeichnung	Größe des Alphabets
binär	2
quinär	5
senär	6
denär	10
7 bit	$2^7 = 128$

Tabelle 2.8: Bezeichnungen für eine Polybios-Matrix

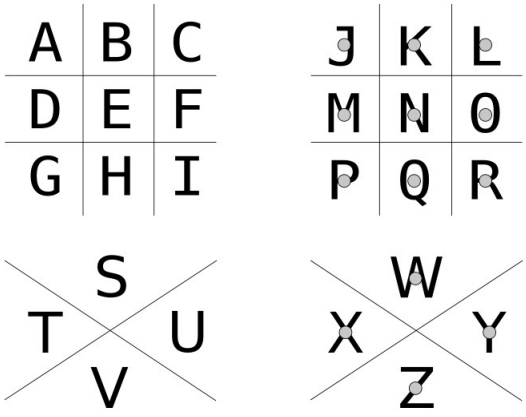
Auf die Frage wieso ADFGX die gewählten Buchstaben der Verschlüsselung sind, antwortet Simon Singh[6], dass jene Buchstaben im Morsecode am deutlichsten zu unterscheiden sind:

· – –
– · ·
· · –·
– – ·
– · ·–

Die Stärke des Algorithmus basiert rein auf der Mehrschichtigkeit des Algorithmus'. Jedoch kann mit der Häufigkeitsanalyse gegen die Substitution gefahren werden (siehe Kapitel Kryptoanalyse) und die Transposition kann durch Durchprobieren entschlüsselt werden. Georges Painvin, einem französischen Leutnant, gelang die entscheidene Entschlüsselung, die einen wesentlichen Beitrag dazu leistete das französische Paris vor der Einnahme durch die Deutschen (nach einem 4-jährigen Kampf) zu schützen. Am 2. Juni hatte er wegen langer Nächte fünfzehn Pfund Gewicht verloren und die Nachricht entschlüsselt. Am Herkunftsort der Nachricht mangelte es an Munition und die Angriffsfront befand sich bereits 80 km vor Paris. Aufgrund des Überraschungsmoments konnten die Deutschen zurückgeschlagen werden und Paris wurde erst 20 Jahre später – unter Hitler – von den Deutschen eingenommen.

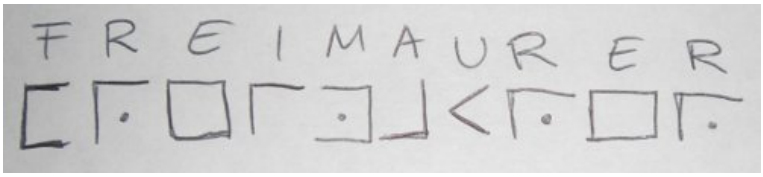
2.5 Freimaurerchiffre

Die Freimaurerchiffre passt zeitlich bereits ins Mittelalter, jedoch handelt es sich ebenfalls um eine monoalphabetischen Substitution. Somit wurde dieser Algorithmus zugleich mit dem Cäsarcode geknackt. Wie die ADFGX-Verschlüsselung wurde der Algorithmus im Zuge der Ideenlosigkeit der Kryptographen des 19. Jahrhunderts populär. Unter den Freimaurern des 18. Jahrhunderts war der Algorithmus für die Kommunikation selbstverständlich.



Die Graphik zeigt zwei Rauten und zwei Kreuze, die mit Buchstaben gefüllt sind. Die

linken Felder unterscheiden sich von den rechten, indem sie keine Punkte in der Mitte besitzen. Um ein Zeichen zo kodieren, müssen wir es in der Graphik suchen und alle Linien bzw. Punkte in der Umgebung notieren. Also werden die Zeichen nur durch ihre Stellvertreter ausgetauscht. Die Häufigkeitsanalyse reicht um die Chiffre zu knacken. Das Wort "Freimaurer" wäre in Freimaurer-Chiffre wie folgt zu kodieren:



2.6 XOR

Auch XOR ("eXclusive OR") lässt sich als eine monoalphabetische Substitution interpretieren. Ein Element a der Menge A wird genau einem Element b der Menge B zugeordnet. Es erfüllt die Bedingungen für eine valide Kodierung (Involutionskriterium). XOR verwendet als Geheim- wie auch Ursprungsalphabet das Minimalalphabet {0,1} (binäre Zahlen) und ist aus der boolschen[3] Algebra bekannt und kann auf ganz einfachen Schaltelementen (Transistorebene) implementiert werden.

Die boolsche Algebra befasst sich mit der Aussagenlogik. Eine Aussage ist wahr, wenn x oder y wahr sind ($x \vee y$). Für die ODER-Operation gilt:

E_1	E_2	A
false	false	false
true	false	true
false	true	true
true	true	true

Tabelle 2.9: ODER in der Aussagenlogik (Wahrheitstabelle)

Ist also Ereignis 1 oder Ereignis 2 oder beides wahr, ist das Ergebnis auch wahr. Einer ähnlichen Systematik folgt XOR ("eXclusive OR"). Es schließt den Fall aus, dass Ereignis 1 *und* Ereignis 2 wahr sind. Oder anders formuliert: XOR ist dann wahr, wenn die 2 Elemente unterschiedlich sind.

Diese Operation hat eine ungewöhnliche Eigenschaft, wenn wir sie auf einen Klartext und einen Schlüssel anwenden. Bei einer ODER-Operation können wir den Ursprungstext nicht mehr rekonstruieren, da die Buchstaben wild durcheinander gewürfelt werden. Die

[3] benannt nach George Boole (1815 – 1864); begründete die moderne Logikaussage auf mathematischer Basis

E_1	E_2	A
false	false	false
true	false	true
false	true	true
true	true	false

Tabelle 2.10: XOR in der Aussagenlogik (Wahrheitstabelle)

1 als Ergebnis tritt häufiger (75%) auf als die 0 (25%). XOR besitzt dieses Verhalten nicht (Verteilung 50:50). Kennen wir ein Element des Klartexts oder Schlüssels, dann können wir aus dem Code wieder den Klartext (oder Schlüssel) errechnen. XOR ist ein Verschlüsselungs- wie auch Entschlüsselungsalgorithmus (wie ROT13).

Klartext	0101011011011001 01
Schlüssel	1010001000110010 01
Code	1111010011101011 00

Tabelle 2.11: Beispiel für eine XOR-Verschlüsselung (0=false, 1=true)

Code	1111010011101011 00
Schlüssel	1010001000110010 01
Klartext	0101011011011001 01

Tabelle 2.12: Entschlüsselung des XOR-Codes

Wir benötigen also nur einen Schlüssel und einen Algorithmus, um die (De)Kodierung durchzuführen. Die XOR-Operation findet eine wunderbar wichtige Anwendung bei Prüfbits. Wir betrachten eine dreidimensionale Matrix. Wir wenden auf jede Zeile XOR an $((z_1 \oplus z_2) \oplus z_3)$. Die resultierenden Element notieren wir in die vierte Spalte. Das selbe machen wir spaltenweise und notieren es als vierte Zeile.

0	1	0	1
1	0	1	0
1	1	1	1
0	0	0	

Wenn wir jetzt eine verfälschte Matrix erhalten, aber die Prüfbits richtig überliefert sind, können wir die Matrix rekonstruieren.

Wir erkennen, dass in der 2. Zeile das 3. Element eine 0 ist. Jedoch ist die Gleichung der Spalte $((1 \oplus 0) \oplus 0 == 0)$ und die Gleichung der Zeile $((0 \oplus 0) \oplus 1 == 0)$ nicht korrekt. Nur

0	1	0	1
1	0	**0**	0
1	1	1	1
0	0	0	

mithilfe der Prüfbits können wir also erkennen in welcher Spalte und Zeile manipuliert wurde. Bei mehreren Änderungen kann eine Gleichung unabsichtlich den Fehler nicht erkennen, aber durch die doppelte Prüfung (Zeile und Spalte) ist die Wahrscheinlichkeit gering. Gefährlich wird es nur, wenn die Prüfbits zerstört werden. Als praxisbezogenes Beispiel können wir die vielen Kratzer auf Datenträgern wie CDs oder DVDs heranziehen. Mithilfe der Prüfbits kann die Software die Fehler wieder bereinigen und die Daten sind nicht fehlerhaft.

Die Sicherheit von XOR ist verschwindend gering. Die ganze Sicherheit beruht auf der Schlüsselgröße. Da es sich aber um ein Minimalalphabet handelt, ist die Wahrscheinlichkeit die richtige Schlüsselziffer für eine Stelle des Codes zu finden, minimal. Verschlüsseln wir drei Buchstaben mit dem Cäsarcode ($26^3 = 17576$) und drei Buchstaben der Menge $\{0, 1\}$ mit XOR ($2^3 = 8$), ist der Cäsarcode sicherer. Allerdings ist dieser Vergleich mehr als nichtig, weil ein geringeres Alphabet zugleich auch mehr Stellen in der Praxisanwendung bedeutet. Ein praktisches Anwendungsgebiet von XOR findet sich in PayTV-Boxen (der Schlüssel ist extrem lang und wird meist monatlich gewechselt).

Wir sprechen hier von Bits, aber Buchstaben sind ja auf Computern intern in Bits gespeichert. Deshalb findet man die XOR-Verschlüsselung erst seit der Verbreitung von Computern. Die XOR-Verschlüsselung findet in dieser Form wenig Anwendung allerdings beziehen sich beinahe alle modernen Algorithmen noch auf XOR (zB DES) und die Wichtigkeit der Prüfbits habe ich bereits erwähnt.

Für Programmierer: Passt bei der boolschen Algebra auf. Es passiert leicht, dass eine Bedingung ($a \ \wedge \ b \vee c$) anders vom Compiler verstanden wird, als ihr es wollt. Vorrangsregeln sind oft etwas eigen. Nutzt Klammern!

2.7 One time pad

Wir haben bei der Substitution gesehen, dass es eigentlich um die Schlüsselwahl geht. Je länger der Schlüssel desto weniger Muster sind im Code erkennbar und die Statistik wird durch den Wind gewirbelt. Schlechte Schlüsselwahl und/oder lange Codetexte sind ein gefundenes Fressen für Kryptoanalytiker. Die Schlüsselwahl sollte nicht die statistischen Eigenschaften der deutschen Sprache übernehmen (keine natürlichen Wörter verwenden!). Den Schlüssellängeansatz verfolgt das One Time Pad.

Wir befassen uns mit der Stärke Nr. 2 der Vigenère-Verschlüsselung:

> Wenn wir jedoch einen langen Schlüssel und kurzen Text verwenden, dann gibt es für (beinahe) jeden Buchstaben ein eigenes Alphabet.

Was ist, wenn Länge des Schlüssels gleich der Länge des Klartexts ist? Dadurch werden die statistischen Angriffe alle ineffizient, weil keine periodischen Verhalten mehr sichtbar sind. Wir kommen zur ersten Verschlüsselung die – theoretisch bewiesen – unknackbar ist.

Die Schlüsselwahl des One Time Pads

Da das One Time Pad sicher ist, benutzten es viele Menschen. Jedoch begehen sie Fehler in der Schlüsselwahl und haben erst wieder keinen zufälligen Schlüssel, der periodische Eigenschaften aufweist. Wenn ich auf der Tastatur eine Kombination zufällig eintippe, passiert es mir automatisch, dass ich zwischen linker und rechter Hand abwechsle. Auch wenn ich das 10-Finger-System nicht anwende, finden wir eine Information vor, die uns die Schlüsselsuche erleichert (wir sprechen dann von pseudozufälligen Schlüsseln). Wir setzen 3 Aspekte für die Schlüsselwahl voraus:

- der Schlüssel ist zufällig
- der Schlüssel wird nur einmalig eingesetzt ("Ephemeralschlüssel")
- der Schlüssel bleibt im Besitz der Kommunikationspartner

2.8 ENIGMA

Eine Tatsache haben die beiden Weltkriege gemeinsam: Die Kriege waren modern. Sowohl in der Luftwaffe wie auch Infanterie gab es wesentliche Fortschritte. Ebenso wurden zum ersten Mal chemische Waffen eingesetzt und ohne moralische Hintergründe zu hinterfragen: Auch die Kryptologie profitierte von den Investitionen.

Die Engima war eine mechanische Maschine, die ein unkompliziertes Verschlüsseln von Nachrichten erlaubte. Sie ist durchaus ebenfalls ins Kapitel "Unkreative Phase des 20. Jahrhunderts" einzuordnen, doch ihre Entschlüsselung war ein Durchbruch beim Benutzen von neuen Kryptowerkzeugen. Es war die erste richtige Kryptomaschine, die eine benutzerfreundliche Verwendung erlaubte und sie brachte außerdem einen der genialsten Köpfe der Informatik hervor.

Die Engima war eine mechanische Maschine. Sie sah ähnlich wie eine Schreibmaschine aus, doch sie besaß einen etwas größeren Körper. Das Tastaturlayout war übrigens das bekannte QWERTZ-Layout. Und damit sollte auch klar sein, dass es eine deutsche Erfindung ist, die während des Weltkriegs in Deutschland eingesetzt wurde. In dem Körper waren Walzen untergebracht, die sich nach jedem Tastendruck weiterdrehten. Durch das

Weiterdrehen wurden ganz neue Schaltkreise gebildet und der nächste Tastendruck wird anders interpretiert als der vorgehende.

Für jeden Tag existierte ein Schlüsselsatz in dem genannt wurde, wie die Walzen zu positionieren sind. Die Verschlüsselung ist durch 4 Faktoren festgelegt:

- Walzenpositionen (120 Positionen)
- Ringstellungen (676 Positionen)
- Grundstellungen (17 576 Positionen)
- Steckerverbindungen (variabel)

Im Laufe der Zeit gab es Weiterentwicklungen. Gegen Ende hin wurde die Angst natürlich größer, dass die Enigma geknackt werden könnte. Besonders die Anzahl der Steckerverbindungen wurden gegen Ende des 2. Weltkrieges auf 10 spezifiziert (0 und 13 möglich). Dadurch waren 150738274937250 (151 Billionen) Steckermöglichkeiten möglich.

Um alle Möglichkeiten auszurechnen, müssen wir das Produkt aller Positionen in dieser Konfiguration bilden[4]:

$$120 \cdot 676 \cdot 17576 \cdot 150738274937250 = 214917374654501238720000$$

Das sind ausgesprochen 214 Quadrillionen ($1Mio^4$) Möglichkeiten die Enigma zu konfigurieren, wenn wir bereits voraussetzen, dass wir 10 Steckerverbindungen erlauben. Diese Schlüsselgröße ist für damalige Verhältnis sehr groß, aber wenn man daran denkt dass zu dieser Zeit zum ersten Mal Maschinen zum Dekodieren entwickelt wurden, erscheint es als logische Entwicklung.

Bei der Verwendung der Enigma kann man 2 große Fehler nennen, die zu ihrem Untergang führte:

- Der Versuch die Enigma involutorisch zu machen
- Die Wiederholung einer Sequenz am Anfang der Nachricht

Aus dem ersten Punkt kann man die Anzahl der Möglichkeiten wesentlich reduzieren. Durch den zweiten Punkt kann man die Stellung der Walzen annähernd erraten.

Die Geschichte und Technik der Enigma fasziniert Menschen seit jeher und es würde jetzt den Rahmen des Dokuments sprengen, sie noch näher zu beschreiben. Wirklich interessant wird die Entschlüsselung. Nach Meinung der Engima-Entwickler kann die maschinell verschlüsselten Klartexte niemand manuell entschlüsseln. Doch in den frühen

[4] frei zitiert nach Wikipedia[9]

Phasen der Enigma war sie noch nicht ausgereift und Marian Rejewski war der erste Kryptoanalytiker, dem ein Einbruch in die Enigma gelang. Er legte mit der Enigma-Gleichung theoretische Grundlagen, um die Enigma zu brechen.

Der Bletchley Park wurde in Folge der wirkliche Geheimort für Entschlüsselungen in Großbritannien. Sämtliche Codes wurden an den Bletchley Park gesandt, um entschlüsselt zu werden. Der BP war zu dieser Zeit schon im Besitz von vielen verschiedenen Maschinen und fertigte selbst Nachbauten an. Insgesamt drei verschiedene Maschinen wurden am BP unterschieden und Alan Turing (1912 - 1954) war ein britischer Mathematiker, der der führende Konstrukteur für Dechiffriermaschinen am BP war. Mit der Turing-Bombe schuff er die Waffe mit der die Enigma geknackt werden konnte und mit der Colossos knackte er die Lorenz-Verschlüsselungsmaschine und schuff den ersten speicherprogrammierbaren Computer.

2.9 Zusammenfassung

Kryptosystem	Jahr	Kryptoanalytische Methode	Gebr.
Skýtala	2500 v.Chr.	Brute-Force	Ja
Cäsarcode	ca. 50. v. Chr.	Häufigkeitsanalyse	Ja
ROT13	20. Jhdt.	Häufigkeitsanalyse	Ja
Vigenère-Code	16. Jhdt.	Kasiski- & Friedmantest, Häuf.	Ja
ADFGVX	ca. 1918	Brute-Force, Häufigkeitsanalyse	Ja
Freimaurer-Chiffre	19. Jhdt.	Häufigkeitsanalyse	Ja
One-Time-Pad	ab 20. Jhdt.	-	Nein
Engima	ab 1918	Brute-Force mit Turing-Bombe	Ja

Kryptosystem	Beruht auf	Kryptograph	Kryptoanalytiker
Skýtala	Transposition	Spartaner	(nicht benannt)
Cäsarcode	monoalphabet. Subst.	Cäsar	(nicht benannt)
ROT13	monoalphabet. Subst.	Internet-User	(nicht benannt)
Vigenère-Code	polyalphabet. Subst.	Vigenère	Babbage
ADFGVX	Subst. und Transpos.	Nebel	Painvin
Freimaurer-Chiffre	Substitution	Freimaurer	(nicht benannt)
One-Time-Pad	polyalphabet. Subst.	Vernam, Mauborgne	(sicheres System)
Engima	polyalphabet. Subst.	(diverse)	Turing, Bletchley Park

Kapitel 3

Kryptoanalyse

Die Kryptoanalyse setzt es sich zum Ziel die Arbeit der Kryptographen nichtig zu machen. Die Kryptographie versucht einen Klartext zu kodieren, sodass es für Menschen nicht mehr möglich auf Anhieb den Inhalt der Nachricht zu verstehen. Die Kryptoanalytiker versuchen den umgekehrten Weg zu gehen und einen Code zu entschlüsseln.

3.1 Der große Erfolg: Häufigkeitsanalyse

Der effektivste Angriff gegen monoalphabetische Verschlüsselung kann mit der Häufigkeitsanalyse vollzogen werden.

Auch wenn die Zeichen gegen Stellvertreter ausgetauscht werden, so übernehmen die Stellvertreter auch deren Eigenschaft der Häufigkeit. Dieses Faktum können wir gegen die monoalphabetische Substitution verwenden. Wir werden feststellen, dass Statistik & Wahrscheinlichkeitsrechnung wohl die wichtigsten mathematischen Instrumente in der Kryptoanalyse sein werden. Jede Sprache hat ihre Eigenheiten. In deutschen Schriften erscheint der Buchstabe "e" mit 17.4% am häufigsten.

In Abbildung 3.1 wurden die Umlaute ä, ö und ü als ae, oe und ue gezählt.

Wenn wir also einen chiffrierten Text wie "Hv zdu hlqpdo" entschlüsseln wollen, können wir feststellen, dass der stellvertretende Buchstabe für e am häufigsten auftreten wird.

Natürlich ist es schwer einen solchen Text zu entschlüsseln. Bei einem solch kurzen Text erinnert die Entschlüsselung eher an Hangman-Spiele als an Kryptologie.

Wir können einmal annehmen, dass es sich um einen deutschen Text handelt. Des Weiteren wird eines der Worte eine Person oder ein Artikel sein. "zdu" könnte für der, die oder das stehen. Der Schlüssel 22 würde ein "z" in ein "d" verwandeln. Der Schlüssel 22 und der Code "d" würden jedoch weder "e", "i" noch "a" ergeben; sondern "h". Deshalb müssen wir annehmen, dass es beim zweiten Wort nicht um einen Artikel handelt.

Buchstabe	Häufigkeit
a	6.51%
b	1.89%
c	3.06%
d	5.08%
e	17.40%
f	1.66%
g	3.01%
h	4.76%
i	7.55%
j	0.27%
k	1.21%
l	3.44%
m	2.53%
n	9.78%
o	2.51%
p	0.79%
q	0.02%
r	7.00%
s	7.27%
t	6.15%
u	4.35%
v	0.67%
w	1.89%
x	0.03%
y	0.04%
z	1.13%
ß	0.31%

Tabelle 3.1: Häufigkeitsverteilung der Zeichen im deutschen Alphabet[8]

Buchstabe	Häufigkeit
h	2
d	2
v	1
a	1
u	1
l	1
q	1
p	1
o	1

Tabelle 3.2: Häufigkeitsverteilung der Zeichen in "Hv zdu hlqpdo"

Für ein deutsches 2-Buchstabenwort kommen nicht viele Variationen in Frage. Ein Beispiel wäre Ei. Da es sich um den Satzanfang handelt, können wir auch Wörter nehmen, die sonst klein geschrieben werden: er, es. Um ein "e" zu erhalten, verwenden wir den Cäsarschlüssel 3. "v" ergibt bei einer Subtraktion mit 3 ein "s". Das erste Wort ist also sehr wahrscheinlich "Es". Es ist natürlich sehr vorteilhaft einen Computer zur Hand zu haben. Wir könnten sogar alle 26 Variationen erzeugen lassen und dann entdecken, dass der Schlüssel 3 tatsächlich die passendsten Wörter kreiiert. "Es war einmal ..." war die berühmte Anfangsphrase der Gebrüder Grimm. Die Häufigkeitsanalyse hat gewonnen, lässt sich bei längeren Texten jedoch besser anwenden.

Die Häufigkeitsanalyse hat aber zwei entscheidene Schwachstellen. Erstens haben wir mehrfach bemerkt, dass die Länge der Nachricht entscheidend ist. Zweitens lässt sich die Häufigkeit verfälschen. Georges Perec verfasste den 200-seitigen Roman "La Disparition" (deutsch "Das Verschwinden" bzw. "Anton Voyls Fortgang"). Jedoch benutzte er kein einziges Mal den Buchstaben e im gesamten Roman. Dem deutschen Übersetzer ist das selbe Meisterwerk gelungen. Der spanische Übersetzer musste den Buchstaben a weglassen, da er in Spanisch der häufigste Buchstabe ist (span. "El secuestro"). Die Häufigkeit lässt sich also manipulieren und nicht immer handelt es sich um eine natürliche Sprache.

3.2 Der Kasiski-Test

Zwar mag Charles Babbage den Vigénere-Code als Erster entschlüsselt haben, jedoch hielt er seine Erkenntnisse geheim und Kasiski konnte seine kryptoanalytischen Methoden ausbauen und damit wesentliche Beiträge zur Kryptoanalyse erbringen. Babbage ging in der Welt der Kryptoanalytiker mit leeren Händen aus, weil er die Erkenntnisse nicht teilte. Kasiski konnte diesen Vorteil nutzen. Beim Kasiski-Test handelt es sich um einen Test, der versucht die Schlüssellänge eines polyalphabetisch verschlüsselten Texts herauszufinden.

Text	`cryptographers are also cryptoanalysts`
Schlüssel	`geekgeekgeekgeekgeekgeekgeekgeekgeekge`
Code	`fsawwphydqilut huf hotp fsawwpbudmaawt`

Man nutzt die Tatsache, dass der Schlüssel sich periodisch verhält und ein Wort im Text eventuell auch wiederholt vorkommt. Der Text wird mit dem selben Schlüssel zweimal verschlüsselt. Aus dieser Tatsache lässt sich folgern, dass der Abstand zwischen diesen beiden auffälligen Text ein Vielfaches des Schlüssels sein muss.

Man muss natürlich bedenken, dass der Code rein zufällig zustande kommen kann. Dies geschieht aber mindestens genauso selten wie durch die Periode. Wenn man also in einem Code zwangig Abstände (mit dem gleichen Faktor) ermittelt hat und drei nicht in dieses System passen (also diesen Faktor nicht besitzen), dann liegt die Wahrscheinlichkeit recht hoch die Schlüssellänge entdeckt zu haben. Versuche immer einen möglichst großen Teiler zu finden. Freue dich also nicht zu früh, wenn du den Faktor 2 gefiltert hast. Und die Schlüssellänge ist natürlich das wichtigste Geheimnis, weil man sonst die Nachricht in (Schlüssellänge) verschiedene Monoalphabete unterteilen kann. Auch hier gilt wieder: Ein großer Schlüssel schwächt die Attacke.

3.3 Der Friedman-Test

Colonel William Frederick Friedman (* 1891 † 1969) veröffentlichte im Jahr 1920 eine Methode, die den Kasiski-Test erheblich erweitert und den sogenannten "Koinzidenzindex" definiert. Die Frage, die er sich stellt: Mit welcher Chance besteht ein zufällig aus dem Text genommenes Buchstabenpaar aus dem selben Buchstaben?

Wir definieren n_1 als die Anzahl der "a"s in einem Text. n_{26} ist dann die Anzahl der "z"s. Wenn wir ein beliebiges "a" aus dem Text (mit n_1 "a"s) auswählen, bleiben $n_1 - 1$ verschiedene andere Möglichkeiten übrig ein "a" zu ziehen. Wir wissen jetzt aus der Mathematik, dass wir mit der folgenden Formel die Anzahl der möglichen Buchstabenpaare (aus "a"s bestehend) ermitteln können:

$$A = \frac{n_1(n_1 - 1)}{2}$$

Wir können diese Formel auf alle Buchstaben anwenden:

$$\frac{n_1(n_1 - 1)}{2} + \frac{n_2(n_2 - 1)}{2} + \ldots + \frac{n_{26}(n_{26} - 1)}{2} = \sum_{i=1}^{26} \frac{n_i(n_i - 1)}{2}$$

Wir wenden jetzt noch die Wahrscheinlichkeitsrechnung an und dividieren "die Anzahl der günstigen Fälle durch die Anzahl der möglichen Fälle" (n ist die Gesamtlänge des

Texts):

$$\kappa = I = \frac{\sum_{i=1}^{26} \frac{n_i(n_i-1)}{2}}{\frac{n(n-1)}{2}} = \frac{\sum_{i=1}^{26} n_i(n_i-1)}{n(n-1)}$$

Kryptologen verwenden das Formelzeichen I (für das Wort Index von Koinzidenzindex), doch Friedman bezeichnete es ursprünglich mit κ und nannte den Test auch "Kappa-Test". Wir können aber noch weiter gehen. Die Wahrscheinlichkeit, dass ein zufällig ausgewählter Buchstabe ein "a" ist, sei p_1 (usw.). Die Wahrscheinlichkeit, dass zwei zufällig ausgewählte Buchstaben zwei "a"s sind, beträgt dann ${p_1}^2$. Die Wahrscheinlichkeit, dass zwei zufällig ausgewählte Buchstaben dem gleichen Buchstaben zugeordnet ist, beträgt also:

$${p_1}^2 + {p_2}^2 + {p_3}^2 + \ldots + {p_{26}}^2 = \sum_{i=1}^{26} {p_i}^2$$

Die Wahrscheinlichkeiten für das Auftreten von Buchstaben sind bereits aus der Häufigkeitsanalyse bekannt. Wenn wir die Wahrscheinlichkeiten für einen deutschen Text einsetzen, erhalten wir den Koinzidenzindex einer zufälligen Buchstabenfolge in einem deutschen Text:

$$I = 0.076059$$

Mit solchen Überlegungen lassen sich zahlreiche Formeln erzeugen, die den Zusammenhang zwischen Buchstaben und Buchstabenpaaren in natürlichen Sprachen dokumentieren. Während sich der Kasiski-Test auf lange Texte spezialisiert, lässt sich der Friedman-Test bereits ab sehr kleinen Texten anwenden. Beide Tests sind eine klassische Kombination in der Kryptoanalyse. Mit dem Friedman-Test kann man des Weiteren näherungsweise feststellen, ob es sich um einen monoalphabetisch oder polyalphabetisch verschlüsselten Text handelt. Wir vergleichen dazu einfach den Koinzidenzindex eines natürlichen Texts mit dem des Codes. Ist der Koinzidenzindex näherungsweise ident, so liegt die Vermutung nahe, dass es sich um monoalphabetische Substitution handelt (durch Ersetzen wird die Wahrscheinlichkeit nicht geändert). Bei einer polyalphabetischen Verschlüsselung wird eine annähernde Gleichverteilung erreicht. Die beiden Tests wurden bei der Vigenére-Entschlüsselung angewandt.

3.4 Le Chiffre indéchiffrable – Kerckhoffs

Auguste Kerckhoffs (* 1835 † 1903) leistete mit seinem Artikel "La Cryptographie militaire" einen wesentlichen Beitrag zur modernen Kryptographie. Er definierte Grundsätze die ein Kryptosystem einhalten muss:

1. Das System muss grundlegend, wenn nicht mathematisch, unknackbar sein
2. Der Diebstahl des Systems durch den Gegner darf nicht Auswirkungen auf die Entschlüsselung haben (keine "security by obscurity")
3. Schlüssel müssen leicht austauschbar und merkbar sein. Kurze Passwörter sollen erlaubt sein, damit man keine Passwörter notieren muss.
4. Kompatibilität mit Telegraphenkommunikation gewünscht
5. Das System muss benutzerfreundlich sein und kein technisches Wissen über das Intera des Kryptosystems voraussetzen
6. Portabilität ist notwendig. Es muss möglich sein, dass nur eine Person das System handhabt
7. Experten sollen das System gut untersuchen

Kerckhoffs sah dies als ein logischer Schluß nach den schlechten Entwicklungen ums 19. Jahrhundert. Statt an neuen Kryptosystemen zu arbeiten, wurden alte - bereits geknackte - kombiniert verwendet, was die Sicherheit nicht wesentlich erhöht.

Gewisse Grundzüge werden wir erkennen können. #1 wurde ursprünglich von allen erfüllt. Experten aus der Kryptoanalyse konnten jedoch immer wieder das Gegenteil beweisen. Deshalb ist dieser Punkt direkt mit Punkt #7 verbunden. #2 hatte fatale Auswirkungen bei der ADFGVX-Verschlüsselung. Im Computerzeitalter werden ständig Laptops beschlagnahmt und die Polizei kann ohne Kenntnis über das Passwort Festplatten nicht entschlüsseln. #3 erfüllen vor allem Hashes (Code von Einweg-Funktionen), die sich bei einer minimalen Änderung des Klartexts sensibelst auf den Code auswirken. #4 entstand infolge der neuen Kommunikationsmittel (Telegraphie verwendet den Morsecode dessen Äquivalent heute der Binärcode ist). #5 war zwar hinreichend für Cäsar erfüllt, doch heute erwartet man, dass man keine Kryptologen zur Ver- und Entschlüsselung benötigt. Heute arbeiten Kryptotools unbemerkt auf unserem Computer. #6 wurde von keinem Kryptosystem gebrochen (jedoch verwendeten Herrscher Boten zur Schlüsselverteilung, was aber nicht im Sinne der Kryptographen war).

Heute spielt besonders eine Regel eine wichtige Rolle und wird dem "security by obscurity" gegenübergestellt:

> Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus' abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels.

Kapitel 4

Moderne Kryptographie – RSA, Hashes und Mathematik

Ich habe erwähnt, dass die Entwicklung von Philologie zu Mathematik statt fand. Ab dem 20. Jahrhundert lassen sich nur mehr Mathematiker als Kryptologen finden und der sprachliche Aspekt bei Verschlüsselungen verschwindet.

Asymmetrischen Verfahren bedienen sich der Mathematik und gehen bereits über die normale Schulmathematik hinaus. Deshalb werden wir der Reihe nach verschiedene Ideen betrachten und sie am Ende auf ein einziges Verfahren anwenden: RSA.

4.1 Das Schlüsselproblem

Egal welches Kryptosystem wir bisher betrachtet haben: Wir haben immer das selbe Problem, welches den Algorithmus recht unbrauchbar macht. Vor allem das One-Time-Pad an sich ist absolut sicher, jedoch muss der Schlüssel ausgetauscht werden. Und dieses Problem ist nach wie vor nicht gelöst. Die bisherigen Lösungen beruhten auf vertrauenswürdigen Kurieren oder/und Steganographie. Die Militärs im Zweiten Weltkrieg waren bereit riesige Geldsumme für diesen Weg der Sicherheit auszugeben. Doch letztendlich leben die Kuriere gefährlich und die Sicherheit ist nicht ausreichend gewährleistet. Mit dem Eintritt in die asymmetrische Verschlüsselung werden wir aber tatsächlich Techniken kennen lernen, die das größte Paradoxon der Kryptologie lösen...

> Schlüssel zu tauschen ohne den Schlüssel zu tauschen.

4.2 Terminologie Teil II

Bob, Alice und Eve 3 Personen, um ein asymmetrisches Kryptosystem zu beschreiben (begründet durch RSA). Alice schickt Bob eine Nachricht und Eve versucht anzugreifen

asymmetrische Verschlüsselung Ver- und Entschlüsselung erfolgt mit unterschiedlichen Schlüsseln

public, private key öffentlicher Schlüssel, der nicht geheim gehalten werden muss, sondern im Gegenteil verteilt werden soll. Privater Schlüssel muss geheim gehalten werden

Signatur Eine Sequenz von Zeichen mit der ein Identitätsnachweis erfolgen kann

modulo Rest einer Division durch 2 Zahlen

ggT Größter gemeinsamer Teiler. zB von 6 und 4 ist es 2.

Primzahlen Zahlen der Menge der ganzen Zahlen, die sich nur durch 1 und sich selbst teilen lassen

Fermats kleiner Satz $a^p \equiv a \bmod p$ mit $p \in \mathbb{P}$

Satz von Euler $a^{\varphi(n)} \equiv 1 \bmod n$ mit $n \in \mathbb{N}$

Faktorisierungsverfahren Stellt eine ganze Zahl in Primfaktorzerlegung dar

q, p Primzahlen

k, k_1, **...** Elemente der ganzen Zahlen $\mathbb{Z}$

n Element der natürlichen Zahlen $\mathbb{N}$

4.3 Diffie-Hellman-Schlüsselaustausch

Whitfield Diffie, Martin Hellman und Ralph Merkle zählen zu den klassischen Computernerds, die ab den 60ern zu finden. Sie lasen jeden Expertenartikel, der mit Mathematik zu tun hatte und gaben niemals auf, eine Lösung zu finden. Wer die Lösung beim ersten Mal nicht findet, findet sie vielleicht beim zweiten Mal. Wer die Lösung beim tausendsten Mal nicht findet, findet sie vielleicht beim tausend-ersten Mal. Genau diese Motivation benötigte man in der Kryptographie um das Schlüsselproblem zu lösen.

> This system faild, but Marty and I continued twisting exponentials around in our minds and discussions trying to make them fit. Marty eventually made the breakthrough early one morning in May 1976. I was working at the

> Stanford Artificial Intelligence Laboratory on the paper that we were shortly to publish under the title "New Directions in Cryptography" 36! when Marty called and explained exponential key exchange in its unnerving simplicity. Listening to him, I realized that the notion had been at the edge of my mind for some time, but had never really broken through.[2]

Whitfield Diffie und Martin Hellman schlugen den sogenannten Diffie-Hellman-Schlüsselaustausch in der Arbeit "New Directions in Cryptography"[3] vor. Ihnen war klar, dass dies eine Revolution sei und gerade Whitfields Biographie verzeichnet eigentlich nur die Frage, wie man das Schlüsselproblem lösen kann. Das ging so weit, dass die NSA ("National Security Agency" der USA) ihn jahrelang verfolgte. Aber er wusste, dass über das Internet (damals noch ARPAnet) Menschen in Zukunft massenhaft Daten austauschen würden und jeder soll sie in Zukunft selbstständig schützen können. Die Privatsphäre muss gewahrt bleiben. Das gab ihm die Motivation trotz staatlicher Behinderung weiterzuforschen. Gemeinsam mit Hellman konnte er seinen Lebenstraum erfüllen: Das Schlüsselproblem zu lösen.

4.3.1 Die Durchführung

Wir definieren drei Zahlen g, a und p, wobei nur die Bedingungen $p \in \mathbb{P}$, $a < (p-1)$ und $g < p$ erfüllt werden müssen. Wäre $a = p-1$, so ergebe sich der kleine Satz von Fermat. Wir würden dann mit einem Fixpunkt verschlüsseln (deshalb schließt man diesen Fall aus). Als Beispiel nehmen wir 2, 5 und 7. p und g sind öffentliche Schlüssel, aber a behalten wir für uns (private key). Wir berechnen nun ...

$$\alpha = g^a \bmod p$$
$$\alpha = 2^5 \bmod 7$$
$$\alpha = 4$$

Wir senden dieses α an Alice, während diese das Verfahren ebenfalls anwendet. Sie kennt g und p, aber da a privat ist, kennt sie es nicht. Sie sucht sich deshalb eine eigene Variable aus $(b < (p-1))$.

$$\beta = g^b \bmod p$$
$$\beta = 2^4 \bmod 7$$
$$\beta = 2$$

Alice sendet ihr β ebenfalls an Bob. Zur Übersicht nochmals:

Nun berechnet Bob:

Person	bekannte Parameter
Bob	g, p, α, β, a
Alice	g, p, α, β, b

$$x = \beta^a \bmod p$$

Und Alice berechnet:

$$x = \alpha^b \bmod p$$

Interessanterweise sind nun diese beiden Werte (x) genau gleich. Mit unseren Werten ergibt das:

$$x = \beta^a \bmod p = 2^5 \bmod 7 = 4$$
$$x = \alpha^b \bmod p = 4^4 \bmod 7 = 4$$

4.3.2 Beweis für das Diffie-Hellman-Verfahren

Wir betrachten die zwei Sätze zur Berechnung:

$$\beta = g^b \bmod p$$
$$x = \beta^a \bmod p$$

β ist der Rest einer Division von g^b durch p. Es gibt also ein ganze Zahl k für die gilt:

$$g^b = k{\cdot}p + \beta$$

Diese Gleichung formen wir nach β um:

$$\beta = g^b - k{\cdot}p$$

Diese Gleichung setzen wir in die 2. Berechnung ein:

$$x = (g^b - k{\cdot}p)^a \bmod p$$
$$x = (g^b - k{\cdot}p){\cdot}(g^b - k{\cdot}p){\cdot}(g^b - k{\cdot}p) \ldots \bmod p$$

Um diese Gleichung zu lösen zu können, müssen wir eine kleine Überlegung machen. Es gilt immer:

$$0 = a \bmod a$$
$$0 = a{\cdot}k \bmod a$$

Es gilt also (da mod$p$):

$$x = (g^b - 0){\cdot}(g^b - 0){\cdot}(g^b - 0) \ldots \bmod p$$
$$x = (g^b){\cdot}(g^b){\cdot}(g^b) \ldots \bmod p$$
$$x = (g^b)^a \bmod p$$
$$x = g^{ba} \bmod p$$

Die selbe Überlegung können wir für α machen:

$$\alpha = g^a - k{\cdot}p$$
$$x = (g^a - k{\cdot}p)^b \bmod p$$
$$x = (g^a - 0)^b \bmod p$$
$$x = g^{ab} \bmod p$$

Wir sehen... die beiden Formeln stimmen komplett überein, wobei wir von unterschiedlichen Parametern ausgegangen sind. Das ist der Beweis, dass das Diffie-Hellman-Schlüsseltaustausch-Verfahren funktioniert.

4.3.3 Problem des Diffie-Hellman-Schlüsselaustauschs

Worin liegt jetzt die Stärke vom Verfahren? Wieso ist es sicher? Eve (Angreifer) würde niemals an a und b rankommen, weil sich die Faktoren nur auf $a{\cdot}b$ reduzieren, wenn der andere auf deinen Exponenten reagiert. Eve könnte einen eigenen Exponenten c definieren und die öffentlichen Schlüssel p und g sind ja bekannt. Es entsteht die Variable γ. Nimmt er jetzt beispielweise α entsteht...

$$x_{invalid} = (g^a - k{\cdot}p)^c \bmod p$$
$$x_{invalid} = g^{ac} \bmod p$$
$$ac \neq ab$$

Das Problem ist direkt verwandt mit dem des diskreten Logarithmus'. Wir kennen beispielweise $g = 89$, $g^a = 704969$ und $g^b = 62742241$. Wie berechnen wir jetzt $a{\cdot}b$? Das ist mit einem handelsüblichen Taschenrechner möglich:

$$\begin{aligned} 89^a &= 704969 \\ \log 89^a &= \log 704969 \\ a{\cdot}\log 89 &= \log 704969 \\ a &= \frac{\log 704969}{\log 89} \\ a &= 3 \end{aligned}$$

$$\begin{aligned} 89^b &= 62742241 \\ b &= \frac{\log 62742241}{\log 89} \\ b &= 4 \end{aligned}$$

Somit ist $a{\cdot}b = 12$.

Jetzt die gleiche Problemstellung, allerdings wissen wir jetzt nur:

$$\begin{aligned} g &= 89 \\ g^a &= 4 \bmod 5 \\ g^b &= 1 \bmod 5 \end{aligned}$$

Und schon wird das Problem nahezu unlösbar. Als sogenanntes Diffie-Hellman-Problem bezeichnet man die Frage:

> Wenn ein Element g und $g^b \bmod p$ sowie $g^a \bmod p$ gegeben sind, welchen Wert hat dann ab?

Die Kommunikation kann also immer nur zwischen zwei Partnern stattfinden und damit erfüllt es den Zweck seiner Sicherheit.

Der Diffie-Hellman-Schlüsselaustausch hat aber genau zwei Probleme.

1. Man-in-the-Middle Eve kann keinen Schaden anrichten, wenn sie Daten der Kommunikation liest. Eve kann jedoch die Pakete α und β abfangen und sie gegen eigene Pakete austauschen. In dem Moment würden die beiden mit Eve kommunizieren. Eve kann dann das Gespräch lenken, wie sie möchte.

   Eve liest Daten der Übertragung: sicher
   Eve schreibt Daten der Übertragung: unsicher

2. Wenn Alice Daten übertragen möchte, muss sie Bob überzeugen an diesem Prozess teilzunehmen und vorher ein paar Daten zu übertragen. Dies sollte solange kein Problem sein wie Bob verfügbar ist. Wenn Bob allerdings in einer anderen Zeitzone wohnt wie Alice, muss Bob zu christlichen Zeiten aufstehen, um einen Schlüssel zur Kommunikation zu erhalten. Das Diffie-Hellman-Schlüsselverfahren verlangt die ständige Bereitschaft beider Kommunikationspartner.

Whitfield Diffie, Martin Hellman und Ralph Merkle hatten dadurch zwar das Schlüsselproblem gelöst, doch hatten sie noch kein sicheres Kryptosystem geschaffen. Whitfield Diffie, der sein Leben lang das Schlüsselproblem zu lösen versuchte, war zwar recht glücklich mit der bisherigen Forschung, fand jedoch Motivation ein besseres zu finden. Er entdeckte, dass das Public/Private-Key-Konzept Erfolg haben könnte. Er formulierte mit Hellman eine mögliche Lösung in "New Directions Cryptograhy", konnte diese Ideen jedoch nicht in ein funktionierendes Kryptosystem bringen. Der Wettbewerb begann. Wer schaffte es am schnellsten Diffie und Hellmans Ideen umzusetzen und ein "sicheres" Kryptosystem zu finden? Ein Jahr später (1976) wurde die Frage beantwortet.

4.4 Public-Key-Verfahren

> We stand today on the brink of a revolution in cryptography

... und sie wussten, wovon sie reden[3]

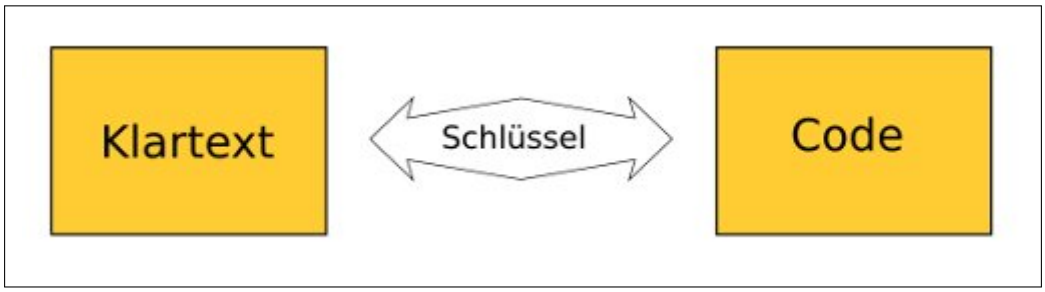


Abbildung 4.1: Symmetrische Verschlüsselung

Bei bisherigen Algorithmen basierte die Geheimniskrämerei auf bloß einem Schlüssel, doch mit dem Public-Key-Verfahren ist der Benutzer im Besitz zweier Schlüssel, die ihm zugewiesen wurden: einen öffentlichen ("public") und einen ("private") Schlüssel. Die Kommunikation erfolgt wie in der Graphik beschrieben.

1. Bob sucht Alice' öffentlichen Schlüssel in einer Kartei (P_{Alice})
2. Bob verschlüsselt seine Nachricht mit Alice' öffentlichen Schlüssel ($C = P_{Alice}(M)$)

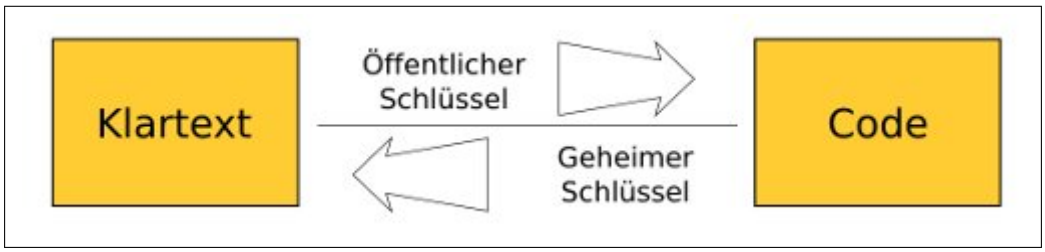


Abbildung 4.2: Asymmetrische Verschlüsselung

3. Bob versendet seine Nachricht an Alice
4. Alice empfängt die Nachricht und entschlüsselt sie mit ihrem privaten Schlüssel ($M = S_{Alice}(C)$)
5. Alice liest die Nachricht
6. Alice verfasst die Antwort und führt die Schritte 1-3 für sich durch.

Dieser kleine Exkurs beschreibt ein utopisches System dessen Implementierung sich niemand vorstellen. Wir haben bereits eine Annäherung mit dem Diffie-Hellman-Schlüsselaustausch erreicht: es gibt einen öffentlichen (g, p) Schlüssel und einen privaten (a, b) Schlüssel. Dennoch ist noch nicht sicher. Das Public-Key-Konzept muss ausbaufähig sein.

Bevor wir uns eine funktionierende/sichere Implementierung näher anschauen, möchte ich die allgemeinen Theoreme der Public-Key-Verfahren nochmals besprechen. Die Public-Key-Bedingung lautet wie folgt:

> Es muss unmöglich sein aus dem öffentlichen den privaten Schlüssel zu extrahieren

Des Weiteren betrachten wir die Vor- und Nachteile:

- Es steht offen ob zur Ent- bzw. Verschlüsselung zwei verschiedene Algorithmen verwendet werden
- Bei neuen Kommunikationsteilnehmern wächst die Anzahl an Schlüsseln langsamer
  symmetrisch: $k = \frac{n(n-1)}{2}$
  asymmetrisch: $k = 2 * n$
  ab $n = 5$ erzeugt asymmetrische Verschlüsselung weniger Schlüssel
- Bei neuen Kommunikationsteilnehmern bleibt der Gesamtaufwand gering
- Es gibt kein asymmetrisches Verfahren welches Schnelligkeit und Sicherheit vereint

- Es ist gutes Schlüsselmanagement notwendig (Kartei, etc.)
- Es gibt eine Signaturmöglichkeit

Eigentlich war es Diffie und Hellman auch nicht ganz klar, ob diese Ideen funktionieren könnte. Sie schlugen das Problem des diskreten Logarithmus zur Implementierung vor, aber konnte es selbst nicht realisieren. Sie gaben aber ein wunderbares – zur Realität analoges – Beispiel mit dem jeder den Gedankengang verstehen kann.

Bob möchte Alice eine Nachricht senden. Deshalb schreibt er sie auf einen Zettel und steckt den Zettel in eine Box. Die Box wird verschlossen und mit einem Vorhängeschloss versehen. Bob kann die Box jetzt jedem beliebigen Postboten überlassen; die Box ist ja versperrt. Wenn sie bei Alice ankommt, kann Alice sie auch nicht öffnen. Sie benötigt den Schlüssel von Bob. Den hat sie aber nicht und deshalb hängt sie ihr eigenes Vorhängeschloss dran. Sie sendet die Box wieder mit der Post zu Bob zurück. Auf der Box befinden sich also momentan zwei Schlösser. Bob entfernt wieder seines und sendet es wieder zurück. Bei Alice angekommen, kann sie ihr Schloss entfernen und die Box öffnen.

Dieses Verfahren beschreibt jetzt zwar nicht das Public-Key-Verfahren, aber gibt eine zündenen Idee, dass eine Realisierung möglich sein könnte. Aber der Gedankengang hat eine Schwierigkeit: Bob verschlüsselt am Anfang, Alice verschlüsselt, Bob entschlüsselt und Alice entschlüsselt auch. Die Reihenfolge scheint durcheinander gewürfelt zu sein. Dabei ist die Reihenfolge sehr entscheidend bei Kryptosystemen und besitzt die Mathematik ein Werkzeug, welches ein solches Verfahren erlaubt? Wie kann man es realisieren? Wenn ich zuerst zu $n = 6$ eine Eins addiere und dann mit 2 multipliziere $(n = (6 + 1)\cdot 3 = 21)$ ist es etwas anderes als wenn ich mit 2 multipliziere und dann Eins addiere $(n = 6\cdot 3 + 1 = 19)$. Die Reihenfolge ist entscheidend.

Diffie und Hellman wussten, dass sie Einwegfunktionen benötigen. Also Funktionen, die sich zwar leicht durchführen lassen, aber die Umkehrfunktion soll sehr schwer sein. Wäre die Umkehrfunktion schwer, so ist sie für den Kommunikationspartner gleich schwer wie für den Angreifer. Deshalb benötigt man Falltürfunktionen: Einwegfunktionen, deren Umkehrfunktion nur leicht durchführbar ist, wenn man eine Zusatzinformation besitzt. Ein Trio fand genau diese Funktion...

4.5 Die GCHQ-Geschichte

Heute weiß man, dass James Ellis, Clifford Cocks und Malcolm Williamson vom GCHQ (Government Communication Headquarter) die Idee des Schlüsselaustauschs knapp 10 Jahre vor Diffie-Hellman zuvor lösten. Im Jahre 1997 wurde bekannt, dass die Kryptologie-Geschichte umgeschrieben werden musste und James Ellis im April 1965 ans GCHQ kam.

Auf Basis des Rauschens[1] entwickelte Ellis den Gedankengang das Rauschen absichtlich zu einer Kommunikation hinzuzufügen. Er konnte damit zeigen, dass es ein Kryptosystem gab, welches den Austausch einer sicher verschlüsselten Nachricht erlaubt, ohne dass die geheimen Schlüssel ausgetauscht werden. Aufgrund dieses Existenzsatzes konnte er zeigen, dass es eine Nadel im Heuhaufen gab, aber nicht wo. Damit war er so weit wie das Diffie-Hellman-Trio (nur 10 Jahre zuvor).

3 Jahre musste das Problem noch warten. 1973 kam der junge Mathematiker Clifford Cocks ins GCHQ zum Team hinzu. Ellis – der bisher immer alleine mit seiner Idee blieb – hatte jetzt einen Partner, der sein Interesse teilte. Cocks hatte wirklich nicht im geringsten Erfahrung im Bereich der Kryptologie, aber als Mathematiker hatte er sich auf Zahlentheorie spezialisiert. Das Erste, was er überlegte, war es somit sich auf Primzahlen und Faktorzerlegung zu konzentrieren. Die Lösung hatte er nach eigenen Angaben nach etwa einer halben Stunde. Nicht nur, dass er das Schlüsselproblem löste; seine wirkliche Arbeit der halben Stunde war das gesamte RSA-Kryptosystem. Jahrelang zerbrach man sich den Kopf darum und Ellis vom GHCQ hatte 3 Jahre lang investiert das Problem zu lösen. Dann erklärt man einem Neuling die Aufgabenstellung und er löst sie in einer halben Stunde.

> Ach, ist ja schön. Man gibt mir ein Problem und ich löse es.[6]

Doch das alles hatte bei weitem nicht solche Folgen wie beim RSA-Trio. Niemand war wirklich an der Lösung interessiert. Ellis war fasziniert von Cocks und Cocks hatte eigentlich keine Ahnung von der Bedeutung der Entdeckung. Schließlich las er nicht so wie Ellis jeden Fachartikel. Ihm wurde zwar am GCHQ gratuliert, aber ihm war das alles so bedeutungslos, dass seine Arbeiten mit der Aufschrift Top Secret versperrt wurden. Praktischen Nutzen konnte das GCHQ auch nicht aus Cocks' Arbeiten gewinnen: die Public-Key-Kryptographie benötigt viel mehr Rechenleistung als symmetrische Verfahren. Eine Implementierung war also damals noch nicht möglich. Cocks' Ideen wurden durch Malcolm Williamson vervollständigt, indem er die mathematische Sicherheit von Cocks Ideen nicht widerlegen konnte. Solange das Faktorisierungsproblem existiert, wird Cocks' Kryptosystem sicher bleiben. Doch es blieb verschlossen. Am GCHQ bleiben Arbeiten Top Secret und so wurde kein Patent beantragt. Man überlegte noch diese "unwichtige" Arbeit zu veröffentlichen (wenn es nicht bringt, schadet es auch nicht), aber bis zum Jahre 2006 blieb diese Geschichte verschlossen.

Was diese Begriffe alles bedeuten (Faktorisierungsproblem, Public-Key, ...), werden wir im folgenden Kapitel betrachten.

[1] Die Übertragung wird durch einwirkende Signale gestört. Da diese Signale zufällig sind, lassen sie sich auch nicht entfernen

4.6 Erzeugung von Pseudozufallszahlen

Um Zufallszahlen zu erzeugen ist ein Algorithmus notwendig, der sehr effizient und schnell eine Vielzahl an Zahlen erzeugen kann. Eine Variante – die diese Forderungen erfüllt – sind die Schieberegister.Wir betrachten ein linear rückgekoppeltes Schieberegister.

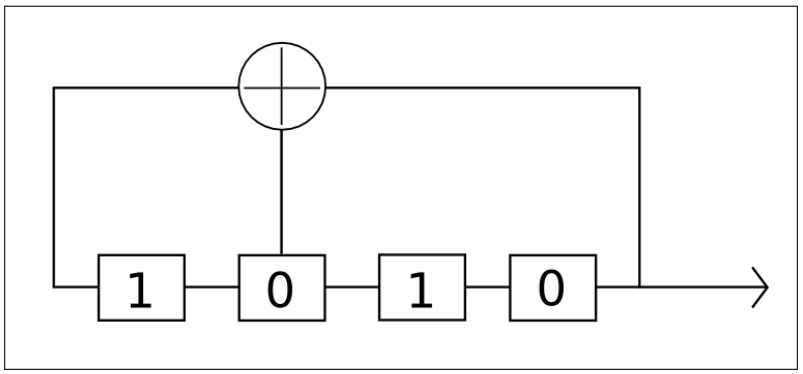


Abbildung 4.3: Schieberegister mit XOR an zweiter Position

Zuerst wird das Register initialisiert (irgendwelche beliebigen Werte gesetzt). Die einzig verbotene Zeichenreihe ist (0,0,0,0), da bei dieser Stellung durch XOR-Operationen kein wahrer Wert entstehen kann und die Reihe bleibt damit immer (0,0,0,0). Die Funktionsweise basiert jetzt darauf, dass die Bits weiterverschoben werden und dabei die Operationen ausgeführt werden. Das linkeste Bit rückt auf die zweite Position. Das zweite Bit rückt auf Position 3 und das dritte Bit auf Position 4. Das vierte Bit geht auf die erste Position und führt vorher eine XOR-Operation mit dem zweiten Wert durch (bevor der erste Wert auf die zweite Position verschoben wird). So verändern sich bei jeder Runde die Zahlen und die Zahlen auf Position 4 nimmt man in einem ungewissen Zeitabstand als Schlüssel. Geheim ist jetzt nur mehr die Initialisierung und auch die Zeit, die man das Schieberegister laufen lässt.

Bei dieser Erzeugung haben wir wieder einen neuen Bereich entdeckt. Nämlich den, der Stromchiffren. Es kann direkt in Echtzeit der Schlüssel erzeugt werden und im Verschlüsselungsalgorithmus eingebaut werden. Der erzeugte Schlüssel eines linear rückgekoppelten Schieberegisters kann durchaus erraten werden, wenn man die Initialisierung. Deshalb kombiniert man Schieberegister. Man könnte zwei Register parallel laufen lassen und nach dem Output des ersten Registers eine bedingte Anweisung anfügen. Wenn der Output von Register 1 gleich 1 ist, dann verwendet man den Output von Register 2, sonst Output von 1. Diese Kombinationsmöglichkeiten lassen sich natürlich auf unendlich viele Dimensionen expandieren. Die Zufallszahl wird schnell und effizient berechnet. Man sollte bloß auf die Wahrscheinlichkeitsrechnung achten, damit man die Auftrittswahrscheinlichkeit eines Wertes nicht erhöht wird.

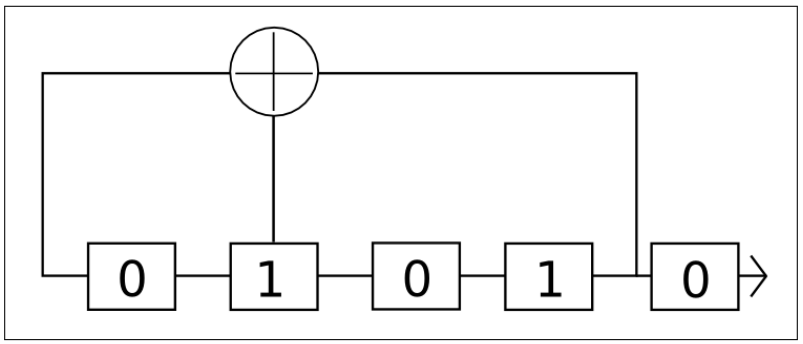


Abbildung 4.4: Schieberegister nach einem Durchlauf

Letztendlich spricht man von Pseudozufallszahlen. Die Problematik liegt daran, dass man mit endlichen Automaten (wie es Computer sind) keine wahren Zufälligkeiten erhalten kann. Auf Dauer gesehen lassen sich immer Perioden erkennen.

4.7 Der Trick mit dem Binomialkoeffizient

Der Binomialkoeffizient ist ein Element (ursprünglich) aus der Kombinatorik. Wenn wir uns fragen, wieviele Möglichkeiten es gibt, k Elemente in einer Menge mit der Mächtigkeit n zu verteilen (die Elements unterscheiden sich nicht), lautet die Antwort $\binom{n}{k}$, wobei dies wie folgt definiert ist...

$$\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!} = \frac{n(n-1)\dots(n-k+1)}{1\cdot 2\cdot 3\dots\cdot k}$$

Wenn wir über Möglichkeiten und Verteilung reden, sprechen wir von ganzen Zahlen und somit gilt $n, k \in \mathbb{Z}$[2]

4.7.1 $(a+1)^p$

Der 1. Satz lautet wie folgt:

$$(a+1)^p = a^p + \binom{p}{1} a^{p-1} + \binom{p}{2} a^{p-2} + \ldots + \binom{p}{p-1} a + 1$$

Wir notieren hierfür einmal $\sum_{i=1}^{4}(a+1)^i$:

[2]Für die Äquivalenz der Definitionen siehe den Beweis auf http://lukas-prokop.at/proj/proof/proof.pdf

$$\begin{aligned}(a+1) &= a+1\\ (a+1)(a+1) &= a^2+2a+1\\ (a+1)(a+1)(a+1) &= a^3+3a^2+3a+1\\ (a+1)(a+1)(a+1)(a+1) &= a^4+4a^3+6a^2+4a+1\end{aligned}$$

Wir bemerken hier eine Dreiecksbewegung. Der Wert am Anfang und Ende ist gleich ($1{\cdot}a^4$ und $1{\cdot}1 \Rightarrow 1$). Bis zur Mitte steigt der Wert und ab der Mitte fällt er wieder. Diese Bewegung kennt man von den Binomialkoeffizienten, weil es gilt:

$$\left(\begin{array}{c} n \\ k \end{array}\right) = \left(\begin{array}{c} n \\ n-k \end{array}\right)$$

Dies kann man über eine erste Definition von Binomialkoeffizienten ganz einfach herleiten:

$$\frac{n!}{k!{\cdot}(n-k)!} = \frac{n!}{(n-k)!{\cdot}(n-(n-k))!}$$
$$\frac{n!}{k!{\cdot}(n-k)!} = \frac{n!}{(n-k)!{\cdot}k!}$$

Aufgrund dieser Eigenschaften besitzen Binomialkoeffizienten diese Dreiecksbewegung. Ok... nun können wir die Reihe oben auch mit Binomialkoeffizienten anschreiben. Zuerst nehme ich $i = 4$ und danach $i = p$.

$$(a+1)^4 = a^4+4a^3+6a^2+4a+1 = \left(\begin{array}{c} 4 \\ 4 \end{array}\right) a^4 + \left(\begin{array}{c} 4 \\ 3 \end{array}\right) a^{4-1} + \left(\begin{array}{c} 4 \\ 2 \end{array}\right) a^{4-2} + \left(\begin{array}{c} 4 \\ 1 \end{array}\right) a^{4-3} + 1$$

$$(a+1)^p = \left(\begin{array}{c} p \\ p \end{array}\right) a^p + \left(\begin{array}{c} p \\ p-1 \end{array}\right) a^{p-1} + \left(\begin{array}{c} p \\ p-2 \end{array}\right) a^{p-2} + \ldots + \left(\begin{array}{c} p \\ 1 \end{array}\right) a^1 + 1$$

Es gibt nur eine Möglichkeit p Elemente in einer Menge mit der Mächtigkeit mit p zu verteilen. Und es gibt p Möglichkeiten ein Element in einer Menge mit der Mächtigkeit p zu verteilen:

$$\left(\begin{array}{c} p \\ p \end{array}\right) = 1 \qquad \left(\begin{array}{c} p \\ 1 \end{array}\right) = p$$

Deshalb gilt:

$$(a+1)^p = a^p + \left(\begin{array}{c} p \\ p-1 \end{array}\right) a^{p-1} + \left(\begin{array}{c} p \\ p-2 \end{array}\right) a^{p-2} + \ldots + \left(\begin{array}{c} p \\ 2 \end{array}\right) a^2 + p{\cdot}a + 1$$

4.7.2 Satz über die Teilbarkeit

Der zweite Satz besagt, dass $\left(\begin{array}{c} p \\ k \end{array} \right)$ mit $p \in \mathbb{P}$ immer durch p teilbar ist.

$$\left(\begin{array}{c} p \\ k \end{array} \right) = \frac{p \cdot (p-1) \ldots (p-k+1)}{1 \cdot 2 \ldots k}$$

Wir müssen konkretisieren, dass $k > p$ für Binomialkoeffizienten keinen Sinn ergibt (man würde mehr Elemente in einer Menge unterbringen als möglich ist) und $k = p$ würde 1 ergeben. Für diese beiden Fälle exkludieren wir den Satz der Teilbarkeit. Also gilt $k < p$ wobei $k, p \in \mathbb{Z}$.

Wir sehen uns die Definition oben an. Die Anzahl der Elemente über dem Bruchstrich ist k. Die unter dem Bruchstrich ist k. Die größeren Zahlen sind stets über dem Bruchstrich zu finden (das Ergebnis ist > 0). Die größte Zahl über dem Bruchstrich p. Deshalb gilt: die größte Zahl aus der Definition des Binomialkoeffizienten ist p. Ist p ein Element der Primzahlen, dann hat p keinen gemeinsamen Teiler mit irgendeiner anderen Zahl, die im Bruch vorkommt. Aus der Definition von Modulo wissen wir:

$$k \cdot a \bmod a \equiv 0$$
$$\frac{k}{a} \bmod a \not\equiv 0$$

Allgemein bleibt eine Zahl nur bei einer Vervielfachung mit einer ganzen Zahl durch sich selbst teilbar. Bei einer Division ist dies nicht der Fall. Ein Binomialkoeffizient wäre nur dann nicht durch p teilbar, wenn der Divisor die (größte) Zahl p beeinflußt. Es ist jedoch sehr interessant, dass Faktoren des Divisors immer durch Kürzungen wegfallen. Diese Kürzungen kommen bei p nicht zustande, weil p keinen gemeinsamen Teiler (1 ausgenommen) mit einer der unteren Zahlen besitzt, weil p eine Primzahl ist. Wir schauen uns ein Beispiel an:

$$\left(\begin{array}{c} 23 \\ 5 \end{array} \right) = \frac{23 \cdot 22 \cdot 21 \cdot 20 \cdot 19}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}$$

21 (aus dem Zähler) ist durch 3 (aus dem Nenner) teilbar. 22 (Zähler) ist durch 2 teilbar. 20 ist durch 4 teilbar. Die resultierende 5 ist durch 5 teilbar.

$$\left(\begin{array}{c} 23 \\ 5 \end{array} \right) = 23 \cdot 19 \cdot 11 \cdot 7$$

Die 23 bleibt jedoch weiterhin unversehrt. Dies liegt daran, dass sie die größte Zahl ist und eine Primzahl ist. Sie wird dadurch nicht durch andere Faktoren verändert.

$$23{\cdot}(19{\cdot}11{\cdot}7) \bmod 23 \equiv 0$$
$$23{\cdot}k \bmod 23 \equiv 0$$
$$p{\cdot}k \bmod p \equiv 0$$
$$a{\cdot}k \bmod a \equiv 0$$

4.8 Modulo

Wir haben Modulo bereits mehrfach erwähnt. Beim Modulo wird der Rest einer Division durch eine Zahl betrachtet. So ist zB der Modulo von 4 durch 2 Null. Oder der Modulo von 15 durch 4 ist 3.

Es wird also so lange die Zahl subtrahiert bis die Subtraktion eine negative Zahl ergeben würde.

$$15 - 4 = 11$$
$$11 - 4 = 7$$
$$7 - 4 = 3$$
$$3 - 4 < 0$$

Also gilt

$$15 \equiv 3 \mod 4$$

Bei dem Gleichheitszeichen mit drei Linien handelt es sich um das Kongruenzzeichen aus der Restklassenrechnung.

4.8.1 Modulo auf Papier

Wer am Papier modulo rechnen möchte, kann den Algorithmus der Division anwenden. Man braucht dazu gar nicht das Ergebnis notieren.

```
225 / 7
 15 / 7
  1 Rest
```

Bei einer Division von 22 durch 7 bleibt 1 Rest ($3{\cdot}7 = 21$). Die Eins notiert man sich in der Zeile drunter und schreibt die "nächste Stelle herab". 15/7 ergibt einen Rest. Also $225 \bmod 7 \equiv 1$.

4.8.2 Symmetrie vs. Mathematik

Was passiert, wenn der "Nenner" kleiner Null wird? Welchen Wert nimmt x an?

$$-2 \equiv x \bmod 3$$

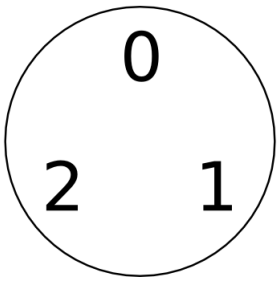


Abbildung 4.5: Ring von Modulo 3

Hier gibt es zwei Auffassungen. Die sogenannte mathematische Variante hat jene Gedankengänge:

- Bei Modulo wird eine Zahl solange abgezogen, bis sie negativ wird. Bei negativem Modulo wird eine Zahl solange addiert, bis sie positiv wird.
- $(-a) \bmod p \not\equiv -(a \bmod p)$
- Wenn die Zahl kleiner wird, dann dreht sich der Moduloring gegen den Uhrzeigersinn.
- $-2 + 3 = 1$
  $-2 \equiv 1 \bmod 3$

Die symmetrische Variante denkt wie folgt:

- Bei Modulo wird der Betrag genommen und dann subtrahiert. Das Vorzeichen wird jedoch vom Ergebnis übernommen.
- $(-a) \bmod p \equiv -(a \bmod p)$
- Wenn die positive Zahl kleiner wird, dann dreht sich der Moduloring gegen den Uhrzeigersinn. Wenn die negative Zahl kleiner wird, dann dreht sich der Moduloring im Uhrzeigersinn.
- $|-2| = 2$
  $-2 \equiv -2 \bmod 3$

Letztenendlich gilt, dass vorwiegend die symmetrische Variante in Programmiersprachen implementiert ist. Dabei wird **immer** die mathematische Variante verwendet. python hat glücklicherweise die mathematische Variante implementiert, aber in Java müsste man eine eigene Funktion schreiben, um die falsche Implementierung zu berichtigen. Für (die Herleitung vom) kleinen Satz von Fermat und im gesamten Dokument brauchen wir die mathematische Variante.

4.8.3 Modulare Arithmetik

Unter modularer Arithmetik versteht man die Wissenschaft, die das mathematische Verhalten von kongruenten Ausdrücken untersucht. Dieses Unterkapitel soll bloß dazu dienen, die wichtigsten Rechenregeln zusammenzufassen.

$$(x+y) \bmod n \equiv ((x \bmod n) + (y \bmod n)) \bmod n$$
$$(x-y) \bmod n \equiv ((x \bmod n) + (-y \bmod n)) \bmod n$$
$$(x \cdot y) \bmod n \equiv (x \bmod n) \cdot (y \bmod n) \bmod n$$
$$x^y \bmod n \equiv (x \bmod n)^y \bmod n$$

$$x \equiv y \bmod m \ \wedge \ r \equiv s \bmod m$$
$$\Rightarrow (x+r) \equiv (y+s) \bmod m$$
$$\Rightarrow (x \cdot r) \equiv (y \cdot s) \bmod m$$

$$x \equiv y \bmod m \ \wedge \ y \equiv s \bmod m$$
$$\Rightarrow x \equiv s \bmod m$$

$$x \equiv y \bmod m \Rightarrow (x+s) \equiv (y+s) \bmod m$$

$$\mathrm{ggT}(m,n) = 1 \ \wedge \ x \equiv y \bmod m \ \wedge \ x \equiv y \bmod n$$
$$\Rightarrow x \equiv y \bmod m \cdot n$$

$$x \cdot y = 0 \bmod p \ \wedge \ p \in \mathbb{P}$$
$$\Rightarrow x \equiv 0 \bmod p$$
$$\Rightarrow y \equiv 0 \bmod p$$

$$\begin{aligned} \mathrm{ggT}(m,x) = 1 \ \wedge \ x{\cdot}y \equiv x{\cdot}s \bmod m \\ \Rightarrow y \equiv s \bmod m \end{aligned}$$

$$a^n \equiv b^n \bmod m$$

4.8.4 Eine weitere Rechenregel

Eine Zahl in einem Modulo-Kreis rotiert scheinbar. Hat sie einen bestimmten Wert erreichen, fällt sie wieder zurück. Eine Rechenregel sagt aus (auch für unser Dezimalsystem[3]), dass wir die Einerstelle von riesigen Zahlen berechnen können, wobei wir selbst nie die riesigen Zahlen ausrechnen zu brauchen.

> 30) Eine Folge ganzer Zahlen ist definiert durch $a_0 = 1$, $a_1 = 2$ und $a_{n+2} = a_n + (a_{n+1})^2$ für $n \geq 0$. Der Rest von a_{2009} bei Division durch 7 beträgt[4]
>
> **A)** 0 **B)** 1 **C)** 2 **D)** 5 **E)** 6

Bei Känguru der Mathematik 2009 selbst schaffte ich das Beispiel nicht, aber daheim mit einem Skript ist die Lösung schnell berechnet.

```
#!/usr/bin/env python

def func(index, a):
    return a[index-2] + (a[index-1])**2

a = [1, 2]
for i in xrange(2, 15):#2010):
    tmp = func(i, a)
    a.append(tmp)
    print i, tmp

print a[2009] % 7
```

Wenn wir mit diesem Programm nach der Antwort suchen, wird unserer Rechner früher heiß laufen, anstatt eine Lösung zu liefern. Wir können uns die Zahlenentwicklung anhand der ersten Werte ansehen (in der linken Spalte stehen die Indizes)

[3] Das Dezimalsystem rechnet einfach mit der Basis 10 und daher wird mit Modulo 10 gerechnet

[4] frei zitiert nach Känguru der Mathematik 2009 vom 23.03.2009

2	5
3	27
4	734
5	538783
6	290287121823
7	84266613096281243382112
8	710086208271835755974856388051748608672870236 7
9	5042224231778729063918929100989070250791737792516107922931 4 384058371278254659634544914 78

Somit passt bereits die achte – der neu berechneten Zahlen – nicht mehr in dieses Dokument. Die 2007. berechnete Zahl wird dementsprechend groß sein und nicht mehr in den Speicher des Computers passen.

Doch wir können einen Trick anwenden. "Nur die Einerstelle der Werte für eine Addition oder Multiplikation sind verantwortlich für die Einerstelle der Summe oder des Produkts". Wir fügen also im Quelltext oben in der Funktion `func` nach dem `return` noch ein Modulo 10 ein, damit die Rückgabewerte kleiner 10 bleiben. Nach dem Ablauf des Programms, nehmen die ersten Variablen (in der Liste `tmp`) folgende Werte an:

2	5
3	7
4	4
5	3
6	3
7	2
8	7
9	1
10	8
11	5

Wie wir beobachten sind es die selben Werte wie oben, aber es sind nur die Einerstellen. Eine Liste mit 2008 Zahlen (die kleiner 10 sind) kann jeder moderne Heimrechner speichern.

2009	1

B) ist die richtige Lösung.

Was lernen wir daraus? Für die Einerstelle des Ergebnisses ist nur die Einerstelle der beiden Summanden bzw. Faktoren verantwortlich. Statt 35 mit 1756 zu multiplizieren, reicht es 5 mit 6 zu multiplizieren, um die Einerstelle des Produkts zu erhalten.

$$35 + 1756 = 1791 \bmod 10 = 1$$

$$5 + 6 = 1$$

Diese Rechenregel lässt sich auf die beiden Punktrechnungsarten (Addition & Multiplikation) anwenden. Nicht jedoch auf Division oder Subtraktion. Der Grund dafür liegt in der Tatsache, dass die Einerstelle bei einer Rechenoperation verantwortlich ist für die Einerstelle des Ergebnisses und alle weiteren Stellen. Die Zehnerstelle (zB eines Summanden) kann jedoch nicht die Einerstelle des Ergebnisses beeinflussen. Nur bei Subtraktion und Division ist dies schon der Fall.

4.9 Potenzieren und Reste berechnen

4.9.1 Square and Multiply in python

"Square and Multiply" ist ein wunderbares Verfahren. Es ist mathematischer Optimierungsalgorithmus, um den Modulo einer potenzierten Zahl zu berechnen. Als ich vor einem mathematischen Problem[5] stand, schrieb ich eine gewöhnliche Anwendung,die $a^c \bmod m$ berechnet. Der Algorithmus benötigt 3 Minuten zur Lösung des Problems. Als ich *Square and Multiply* anwand, verkürzte sich die Laufzeit auf 7 Sekunden. Das liegt daran, dass bei einem normalen Verfahren, der Computer für a^c $(a{\cdot}(c-1))$ Multiplikationen durchführen muss. Bei SaM reduziert es sich auf die Anzahl der Stellen der Binärzahl von c. Im Vergleich bedeutet dies bei der Rechnung 12334^{8402} herkömmlich 8402 Schritte und mit SaM nur 14 Schritte.

In modernen Programmiersprachen wie python ist SaM bereits eingebaut (in der Funktion pow()). SaM vereinfacht das Potenzieren, indem der Exponent mit den Potenzrechenregeln zerlegt wird. Es gibt verschiedene Varianten. Normalerweise arbeitet man mit der binären Exponentation. Der folgende Programmcode kommt ohne Binärzahlen aus.

```
a = 12334
c = 8402
m = 13

res = 1
while c != 0:
    while c % 2 == 0:
        c = c / 2
        a = (a**2) % m
    c = c - 1
    res = (res * a) % m
```

[5] http://projecteuler.net

```
return res

print res
```

4.9.2 Square and multiply am Papier

Wer Square and Multiply am Papier rechnen möchte, braucht die Potenz als Binärzahl. Das Umrechnen kann mit einer Tabelle erfolgen. Ich führe hier die Rechnung 123^{84} mod 13 durch. Das Verfahren kann man ohne Modulo durchführen, indem man den Modulo auch bei den Rechnungen weglässt.

	64	32	16	8	4	2	1
84	1	0	1	0	1	0	0

Wie ist diese Zahl zustande gekommen? $12 = 1{\cdot}10^1 + 2{\cdot}10^0 = 12_{10}$ und für binäre Zahlen gilt äquivalent $12 = 1{\cdot}2^3 + 1{\cdot}2^2 + 0{\cdot}2^1 + 0{\cdot}2^0 = 1100_2$. In der Tabelle habe ich die größte Zahl 2^x angeschrieben, die kleiner als 84 ist. Unter diese Zahl kommt eine Eins. Danach folgt die nächstkleinere (2^5). $64 + 32 > 84$ und deshalb notiert man drunter eine Null. Nach diesem System rechnet man bis 2^0 und erhält dann die binäre Zahl. Durch solche Tabellen kann man binäre Zahlen sehr gut umrechnen. Für 84 gilt also $86_{10} = 1010100_2$.

Wir nehmen die binäre Zahl und entfernen alle führenden Nullen (falls wir welche angeschrieben haben). Danach entfernen wir noch die erste 1. Wir ersetzen alle 1en mit "QM" und 0en mit "Q". Als Ergebnis erhalten wir einen Code wie "QQMQQMQQ". Dies verwenden wir jetzt als Anweisung (Q = quadrieren und M = Multiplizieren). Nach jeder Operation führen wir modulo 13 aus.

$$(((((((123)^2)^2){\cdot}123)^2)^2){\cdot}123)^2)^2$$
$$(((((((10)^2){\cdot}123)^2)^2){\cdot}123)^2)^2$$
$$(((((9{\cdot}123)^2)^2){\cdot}123)^2)^2$$
$$((((2^2)^2){\cdot}123)^2)^2$$
$$\dots$$
$$1$$
$$123^{84} \bmod 13 = 1$$

Durch dieses Verfahren ist die Potenz 2 notwendig und die Multiplikation mit der Basis. Beides ist mit der Hand problemlos durchzuführen. Und der Computer benötigt nur so viele Schritte, wie lang die Binärzahl ist. Sonst benötige er 83 Multiplikationen...

$$123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}123{\cdot}\ldots$$

4.10 Kongruenz

Zwei Zahlen sind genau dann kongruent zueinander, wenn deren Rückgabewert nach einer Modulo-Berechnung den selben Wert ergeben. Zwei Werte a und b sind also kongruent zueinander, wenn gilt:

$$a \bmod n = b \bmod n$$

Wann sind solche Zahlen kongruent zueinander? Wenn n den Betrag der Differenz von $a - b$ teilt. Teilen bedeutet modulo 0. Ein Zahlenbeispiel wäre...

$$13 \bmod 6 = 19 \bmod 6$$
$$1 = 1$$
$$|(a - b)| \bmod n = 0$$
$$|(13 - 19)| \bmod 6 = 0$$
$$6 \bmod 6 = 0$$

wahre Aussage

Eine Beziehung $a \bmod n = x$ lässt sich auch wie folgt anschreiben:

$$a = k{\cdot}n + x$$

Man schreibt formal korrekt an (beachte, dass das mathematische Identitätszeichen mit drei Strichen verwendet wird):

$$a \equiv b \bmod n$$

4.11 Eulersche Funktion

Die Eulersche Funktion gibt die Anzahl teilerfremden Zahlen kleiner dem Parameter zurück. Teilerfremd ist eine Zahl, wenn sie mindestens 2 Teiler mit einer anderen Zahl besitzt. Die Zahl 12 und die Zahl 1 haben nur die 1 als gemeinsamen Teiler und sind deshalb teilerfremd zueinander. 12 und 2 besitzen 1 und 2 als gemeinsamen und sind

daher nicht teilerfremd. 12 und 3 besitzen 1 und 3 als Teiler. Wenn wir ständig so weitermachen, werden wir entdecken, dass nur 1, 5, 7 und 11 die 1 als gemeinsamen Teiler mit 12 besitzen. Die Anzahl der teilerfremden Zahlen kleiner dem Parameter 12 ist also 4.

Für Primzahlen gilt eine besondere Formel:

$$\varphi(n) = (n-1); \qquad n \in \mathbb{P}$$

Dies ergibt sich daraus, dass nur 1 ein gemeinsamer Teiler mit 1 ist und sonst keine Zahl kleiner der Primzahl.

4.11.1 Berechnung der Eulerschen Funktion

Der bekannteste Algorithmus für Informatiker ist der Euklidische Algorithmus. Er berechnet den größten gemeinsamen Teiler zweier Zahlen. Der Algorithmus geht – entgegen der Namensgebung – nicht auf Euklid als Entdecker zurück, wurde jedoch von ihm in dem Werk "Les Elements" erwähnt.

```
def euklid(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def euklid_recursive(a, b):
    if b == 0:
        return a
    else:
        return euklid_recursive(b, a % b)
```

Im Programmcode wurde der Algorithmus zuerst iterativ, dann rekursiv implementiert.

Gut... mit diesem Algorithmus können wir den ggT zweier Zahlen berechnen. Aber was bringt uns das bei der Eulerschen Funktion? Ganz einfach müssen wir alle gemeinsamen Teiler testen. Ist der Teiler größer 1, dann wissen wir, dass es mindestens 2 Teiler gibt und die Zahl somit teilerfremd ist.

```
num = 13
prime = True
for i in xrange(num):
    if euklid(num, i) != 1:
        prime = False
return prime
```

4.11.2 Ein Satz zur Eulerschen Funktion

Die eulersche Funktion gibt die Anzahl der Zahlen kleiner n zurück, die teilerfremd zu n sind. Das heißt für 11 gilt:

$$\varphi(11) = |\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}| = 11 - 1 = 10$$

Wie sieht es jetzt bei einer Multiplikation von Primzahlen aus?

$$\varphi(11{\cdot}13) = 120$$

Wenn wir genau überlegen, dann ergibt sich die 120 ebenso aus...

$$\varphi(11{\cdot}13) = 10{\cdot}12 = 120$$

Woran liegt das? Wir können es ganz einfach herleiten.

Die maximale Anzahl der Zahlen kleiner n und teilerfremd zu n, kann maximal $(n-1)$ sein.

$$\varphi(p{\cdot}q) = p{\cdot}q - 1 \ldots$$

p und q haben keine Teiler, aber besitzen Vielfache im Bereich der Differenz der höheren von der niedrigeren Zahl. Wir müssen also alle Vielfachen von p abziehen:

$$Q = \{q, 2q, 3q, 4q, \ldots (p-1)q\}$$
$$|Q| = (p-1)$$

Das letzte Element kleiner $p{\cdot}q$ ist $(p-1){\cdot}q$ bzw. $p{\cdot}(q-1)$. Deshalb ist es das letzte Element dieser Folge. Die Größe der Menge Q ist $(p-1)$. Für die Vielfachen von q machen wir den selben Prozess.

$$P = \{p, 2p, 3p, 4p, \ldots (q-1)p\}$$
$$|P| = (q-1)$$

Diese Mengen müssen wir von $pq - 1$ abziehen:

$$\varphi(p{\cdot}q) = p{\cdot}q - 1 - |P| - |Q|$$
$$\varphi(p{\cdot}q) = p{\cdot}q - 1 - (q-1) - (p-1)$$

Jetzt brauchen wir nur mehr termumformen. Wir heben p heraus:

$$\varphi(p{\cdot}q) = p{\cdot}q - 1 - q + 1 - p + 1$$
$$\varphi(p{\cdot}q) = p{\cdot}q - p - q + 1$$
$$\varphi(p{\cdot}q) = p(q-1) - (q-1)$$
$$\varphi(p{\cdot}q) = (p-1)(q-1)$$

Wir folgern daraus: Die Anzahl der Zahlen kleiner dem Produkt von zwei Primzahlen (p, q), die teilerfremd zum Produkt sind, ist $(q-1)(p-1)$.

4.12 Theorie der Primzahlen

> Unter Primzahlen versteht man alle Zahlen, die nur durch sich selbst und 1 teilbar sind

Primzahlen haben sehr interessante Eigenschaften. Es gibt verschiedene Ansätze, wie man zu Primzahlen gelangt. Einen Ansatz stellt Rudolf Taschner[7] vor.

Wir möchten alle Zahlen finden, um alle Zahlen durch eine Multiplikation abbilden zu können. Mit 1 wären wir erfolgreich. $1 * a = a$ wobei a für jede beliebige Zahl $a \in \mathbb{R}$. Aber 1 ist nicht valid. Die Griechen sahen die Eins nicht einmal als richtige Zahl an. Eins war die Einheit. Wir suchen also alle anderen natürlichen Zahlen. Wir fügen in unsere Liste einmal 2 hinzu. 3 lässt sich nicht durch 2 darstellen, also fügen wir es hinzu ([2 ,3]). 4 lässt sich durch 2 darstellen. 5 lässt sich nicht durch 2 oder 3 darstellen ([2, 3, 5]). 6 durch 2. 7 wieder nicht ([2, 3, 5]). 8 durch 2. Das geht jetzt immer so weiter. Wir können einmal alle geraden Zahlen ausschließen. Damit wir weitere Zahlen finden können wir einmal beobachten, wann ungerade Zahlen bei einer Multiplikation auftreten.

gerade * gerade	gerade
ungerade * gerade	gerade
gerade * ungerade	gerade
ungerade * ungerade	ungerade

Tabelle 4.1: Multiplikationsregel für (un)gerade Zahlen

Wir können es etwas anders interpretieren. Die geraden Zahlen sind ja jene Zahlen, die sich durch modulo 2 gleich Null kennzeichnen.

$$0 \bmod 2 = 0$$
$$1 \bmod 2 = 1$$
$$2 \bmod 2 = 0$$

$$3 \bmod 2 = 1$$

$$16 \bmod 2 = 0$$

$$17 \bmod 2 = 1$$

Und wenn wir jetzt die Additions- und Multiplikations-Tabelle für die modulo 2 notieren, erhalten wir einen schönen Überblick:

+	0	1
0	0	1
1	1	0

Tabelle 4.2: Additionsregeln für modulo 2

*	0	1
0	0	0
1	0	1

Tabelle 4.3: Multiplikationsregeln für modulo 2

Die Additionstabelle habe ich deshalb notiert, weil sich die Ähnlichkeit zu XOR aus Kapitel 2.6 nicht leugnen lässt. Wir können formulieren: Modulo 2 hat das selbe Verhalten wie XOR.

Den Primzahlen ordnet man auch inoffiziell das Mengensymbol $\mathbb{P}$ zu.

Zurück zu unserer ursprünglichen Frage: Wie kann man alle natürlichen Zahlen erhalten, mit deren Multiplikation sich alle Zahlen abbilden lassen? Wir brauchen also nur ungerade Zahlen betrachten, weil eine Multiplikation mit einer geraden Zahlen ergibt ersichtlich immer eine gerade Zahl. Nun betrachten wir beispielweise 15. 15 ist eine ungerade Zahl (da $15 \bmod 2 = 1$), aber keine Primzahl. Das liegt daran, dass sie das Produkt der beiden (zufällig) Primzahlen 3 und 5 ist. Die Hälfte der Zahlen haben wir bereits ausgeschlossen und zusätzlich suchen wir nach Produkten. Nach Produkten von allen Zahlen brauchen wir gar nicht suchen (siehe Tabelle 4.3). Bevor wir jetzt weiterdenken bemerken wir, dass wir hier von Primzahlen sprechen. Mit Primzahlen lassen sich alle natürlichen Zahlen durch Multiplikation (oder durch sich selbst) darstellen.

$$2, 3, 5, 7, 11, 13, 17, \ldots$$

An der Stelle fassen wir die Punkte nochmals verallgemeinert zusammen:

- Primzahlen lassen sich nur durch 1 und sich selbst teilen
- Mit Primzahlen lassen sich alle natürlichen Zahlen abbilden

- Je größer der Zahlenbereich wird, desto seltener treten Primzahlen auf

Aus dem zweiten Satz folgt automatisch die Überlegung der Primfaktorzerlegung. Eine Zahl können wir (beinahe eindeutig) in Primzahlen zerlegen. Die Zahl 6 lässt sich beispielweise nur als $2 * 3$ anschreiben. Falsch, denn auch $2 * 2 * 2$ ist 6, doch wir versuchen den Term möglichst kurz zu halten, indem wir jede Primzahl nur einmal verwenden.

> Unter Primfaktorzerlegung versteht man die Zerlegung einer Zahl als Produkt seiner Primfaktoren

Jetzt stellen wir uns 3 Fragen. Diese müssen wir näher betrachten, damit wir Primzahlen zur Verschlüsselung anwenden können.

- Kann man hervorsagen, wann die nächste Primzahl auftreten wird?
- Ist eine Zahl x eine Primzahl?
- Wie können wir alle Primzahlen erzeugen?

4.12.1 Euklid's Beweis für die unendliche Anzahl an Primzahlen

Wir wissen:

> Alle ganzen Zahlen lassen sich als Produkt von Primzahlen abbilden

Euklid definiert eine Menge:

$$N = p_1 \cdot p_2 \cdot p_3 \cdot p_r + 1; \qquad p_i \in \mathbb{P}$$

Eine Division dieser Menge durch eine der Primzahlen ergibt immer den Rest 1. Wieso?

$$a \equiv b \bmod c; \qquad a \geq b$$

Das $a \geq b$ sollte nur klar machen, dass a die ursprüngliche Zahl ist und ein unbekannter Wert k so oft abgezogen wird, bis nur mehr b übrig bleibt. Wir können die Modulo-Operation umschreiben:

$$\Rightarrow k \cdot c + b = a$$

Ok... halten wir dies fest und kehren zu Euklid's Menge zurück. Wir nehmen die Aussage als wahr an. Dann gilt:

$$p_1{\cdot}p_2{\cdot}p_3 \ldots p_n + 1 \equiv 1 \bmod p_n; \qquad n \in \mathbb{Z}$$
$$\Rightarrow k{\cdot}p_n + 1 = p_1{\cdot}p_2{\cdot}p_3 \ldots + 1$$

Wenn wir annehmen, dass n beispielweise 2 ist, dann gilt:

$$k = p_1{\cdot}p_3{\cdot}p_4 \ldots$$

Wir sehen, dass k eine ganze Zahl ist, was wir vorher erwarteten. Mit k wird sozusagen der Rest der Gleichung ausgeglichen. Wir haben bewiesen, dass die Division von Produkt von n Primzahlen plus Eins durch eine der Primzahlen immer den Rest 1 hinterläßt.

Zurück zum Beweis:
Da die Zahl den Rest 1 hinterlässt, ist sie nicht als Produkt von bisherigen Primzahlen darstellbar. Deshalb muss sie als eine neue Primzahl definiert werden, die nur 2 Teiler (1 und sich selbst) hat. Diese Aussage lässt sich auf jede beliebig große Menge an Primzahlen anwenden und Euklid sah darin den Beweis für die Unendlichkeit der Primzahlen.

Noch größere Primzahlen?
Die Frage lautet, wieso man diese Verfahren nicht verwendet, um noch mehr Primzahlen zu erzeugen. Das Problem ist, dass es nicht alle Primzahlen erzeugt, wobei das Verfahren selbst die Kenntnis von allen bisherigen Primzahlen vorher sieht.

4.12.2 Wozu Primzahlen?

Primzahlen haben eine wunderbare Eigenschaft: Sie sind unvorhersehbar. Dies sehen wir auch, wenn wir die Frage Nummer 1 (”Kann man hervorsagen, wann die nächste Primzahl auftreten wird?” auf Seite 63) zu analysieren versuchen.

	0	1	2	3	4	5	6	7	8	9	10
$2{\cdot}x$	0	2	4	6	8	10	12	14	16	18	20

Tabelle 4.4: Tabelle mit $2{\cdot}x$

Man benötigt nicht viel Gehirnschmalz, um die nächste Zahl (22) hervorzusagen. Genauso ist es nicht schwer diese Folge zu notieren und die Ausgangsfunktion zu erraten.

Die Zahl Null ist eine ganz böse Zahl. Einerseits ist sie als Faktor total untauglich und andererseits können wir den Modulowert erahnen, wenn wir einen Wert mit 0 erhalten. Nehmen wir zum Beispiel die $2{\cdot}2$ heraus (3. Zeile und 4. Spalte).

$$2{\cdot}2 \bmod x = 0$$
$$4 \bmod x = 0$$

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	0
2	0	2	0	2	0
3	0	3	2	1	0
4	0	0	0	0	0

Tabelle 4.5: Multiplikationstabelle mit Modulo 4

Mögliche Lösungen sind 2 und 4. Damit können wir die Ausgangsfunktion näherungsweise erraten. Wenn wir weitere Werte besitzen, entdecken wir, dass es sich um Modulo 4 handelt (zB da $1 \cdot 2 \bmod x = 2$).

Was wir jetzt wollen ist eine Folge von Zahlen, deren nächster Wert unvorhersehbar ist und möglichst wenig Nullen enthält. Hier kommen die Primzahlen ins Spiel. Bei 4 handelt es sich um keine Primzahl, aber wir schauen uns einmal die Tabelle mit Modulo 7 an. Die Zeile und Spalte mit 0 als Faktor lassen wir weg, da wir gesehen haben, dass sie nur böse Zahlen erzeugt.

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	0
2	2	4	6	1	3	5	0
3	3	6	2	5	1	4	0
4	4	1	5	4	6	3	0
5	5	3	1	6	4	2	0
...							

Tabelle 4.6: Multiplikationstabelle mit Modulo 7

Ok... hier sehen wir bereits: Die böse Zahl kommt nur mehr in der Spalte mit 7 vor. Dies ist leider unvermeidbar. Aber wir können die Funktion noch komplexer machen, damit die Zahlen noch unerwarteter auftreten.

	1	2	3	4	5	6	7
$x^3 \bmod 11$	1	8	5	9	4	7	2

Tabelle 4.7: Multiplikationstabelle mit x^3 mod 11

Das heißt, wenn die Modulozahl eine Primzahl ist, kommen selten Nullen vor (erst bei $11^3 \bmod 11$ wieder und nur 45mal bei $x < 500$), was wesentlich vorteilhafter ist und die Sicherheit des Kryptosystems erhöht. Neben Null ist 1 auch böse, weil man dadurch zwar nicht den Modulowert erraten kann, aber der Faktor 1 trotzdem den Wert nicht

kodiert. Null und Eins bezeichnet man deshalb als *Fixpunkt* (Werte, die einen Faktor nicht sauber verschlüsseln).

4.12.3 Sieb des Eratosthenes

Auf die Frage, wie man **alle** Primzahlen erzeugen kann, wusste Eratosthenes von Kyrene (* 276 v. Chr. † 194 v. Chr.) eine Antwort. Wir könnten natürlich einen Algorithmus programmieren, der nach oben wandert und dabei berechnen wir für jede Zahl die Anzahl an Teilern. Ist sie bloß 2 oder kleiner (1 und sich selbst), so ist es eine Primzahl. Doch dieses Verfahren wäre viel zu aufwändig. Eratosthenes dachte sich den umgekehrten Weg: Er nimmt an, alle natürlichen Zahlen sind Primzahlen. Er nimmt jetzt einen Teiler her und streicht alle Zahlen, die jenen Teiler haben. Dadurch wird eine enorme Geschwindigkeitssteigerung erzielt.

```
def sieve_of_eratosthenes(limit):
    sieve = [False, False] + [True] * (limit - 2)
    result = 0
    next = 2
    limit2 = sqrt(limit)

    while next < limit2:
        for i in xrange(next*2, limit, next):
            sieve[i] = False
        next += 1
    return sieve
```

Lassen wir diese python-Implementierung bis 100 laufen, erhalten wir eine Liste mit 100 Zahlen. Jeder Zahl ist ein boolscher Wert angehängt, ob die Zahl eine Primzahl ist.

Wieso läuft das Sieb des Eratosthenes bis $\sqrt{n}$?
Die letzte (größte) Zahl – die überprüft werden muss – ist n. n entsteht durch die Multiplikation von $\sqrt{n}\cdot\sqrt{n}$. $(\sqrt{n}+1)\cdot\sqrt{n}$ ist eine Zahl über n und $(\sqrt{n}-1)\cdot\sqrt{n}$ ist unter n. Das bedeutet, wenn innerhalb des Bereichs um 10 bleiben wollen, müssen wir den einen Faktor anheben und den anderen erniedern. Da das Kommutativgesetz für ganze Zahlen gilt, spielt es keine Rolle, welcher Zahl wir welche Rolle zuordnen.

$$100 \approx 10{\cdot}10, 9{\cdot}11, 8{\cdot}12, 7{\cdot}14, 6{\cdot}16, 5{\cdot}20, 4{\cdot}25, 3{\cdot}33, 2{\cdot}50, 1{\cdot}100$$

Und aufgrund dieser Reihe gilt, dass alle großeren Faktoren (beispielweise 16 von 6·16) bereits durch den anderen Faktor abgedeckt sind (6). Es werden keine Zahlen nach 10 mehr auftreten, die neue Faktoren zudeckt. Beispielweise 11 deckt 22, 33, 44, 55, 66, 77, 88 und 99 zu. Dazu benötigt es aber die Faktoren 2, 3, 4, 5, 6, 7, 8 und 9, deren Vielfachen aber bereits zugedeckt wurden. 11·11 wäre der erste Faktor, den 11 neu zudecken würde, aber der ist größer als $10{\cdot}10 = 100$.

4.13 Probedivision

Bei der Probedivision handelt es sich um Algorithmus, der sowohl ein Primzahltest als auch Faktorisierungsverfahren ist.

```
#!/usr/bin/env python

import math
import sieve

def divisible(a, b):
    if (a / b) * b == a:
        return True
    else:
        return False

num = int(raw_input('Gib mir eine ganze Zahl: '))

primes = sieve.sieve_of_eratosthenes(num)
factors = []
end = math.ceil(math.sqrt(num))
p = 2
i = 0

while num > end:
    if divisible(num, primes[i]):
        factors.append(primes[i])
        num = num / primes[i]
    else:
        i += 1

print factors
```

4.14 Modulare Inverse

$$a \equiv 1 \bmod n \Leftrightarrow a = k{\cdot}n + 1$$

$$5 \equiv 1 \bmod 4 \Leftrightarrow 5 = k{\cdot}4 + 1$$

$$e{\cdot}d \equiv 1 \bmod \varphi(N) \Leftrightarrow 1 = e{\cdot}d + \varphi(N){\cdot}k$$

Beim RSA-Verfahren haben wir die Aufgabenstellung $e{\cdot}d \equiv 1 \bmod \varphi(N)$. Diese ist (wie wir oben sehen) äquivalent zur Aufgabe $1 = e{\cdot}d + \varphi(N){\cdot}k$. Und diese Problemstellung

kennt die Mathematik bereits und liefert einen Algorithmus, der die Gleichung lösen kann: der erweiterte euklidische Algorithmus (den einfachen Euklid haben wir bereits in Sektion 4.11.1 kennen gelernt).

Satz der Vielfachsummendarstellung:

> Sei d der größte gemeinsame Teiler der Zahlen a und b. Dann gibt es ganze Zahlen x und y mit der Eigenschaft, dass
>
> $$d = ax + yb$$

Erweiterter Euklidischer Algorithmus:

Berechne $1 = 71{\cdot}d + 8{\cdot}k$.

$$71 = 8{\cdot}8 + 7$$
$$8 = 1{\cdot}7 + 1$$
$$7 = 7{\cdot}1 + 0$$

Erhalten als Modulo-Wert (vierter Wert) 0 so beenden wir diese Rechnung. Aus der letzten Zeile können wir denn $\mathrm{ggT}(71, 8)$ ablesen (dritter Wert). Wichtig ist, dass wir das System bei der Rechnung stets weiterführen.

$$\mathrm{ggT}(71, 8) = 1$$

Um jetzt unsere ursprüngliche Aufgabe zu erledigen, müssen wir sozusagen zurück rechnen.

$$1 = 8 - 1{\cdot}7$$
$$1 = 8 - 1{\cdot}(71 - 8{\cdot}8) = 1{\cdot}8 - 1{\cdot}71 + 8{\cdot}8 = -1{\cdot}71 + 9{\cdot}8$$

Und schon haben wir eine Gleichung, die unsere Aufgabenstellung erfüllt. Zugleich ist es auch die Lösung, die mit den kleinsten Werten, die Gleichung erfüllt.

$$d = -1; k = 9$$

4.15 Fermat's kleiner Satz

$$a^p \equiv a \bmod p \qquad a \in \mathbb{Z}; p \in \mathbb{P}$$

Wir wollen jetzt den kleinen Satz von Fermat (kurz: Kleiner Fermat) beweisen. Dazu führen wir eine vollständige Induktion durch. Das bedeutet wir zeigen, dass die Aussage für die erste natürliche Zahl 1 gilt (Induktionsanfang). Danach zeigen wir, dass – wenn es für die Variable k gilt (Induktionsannahme) – ebenso für $k+1$ gilt (Induktionsschritt).

Induktionsanfang:

$$1^2 \equiv 1 \bmod 2$$

wahre Aussage

Induktionsannahme:

$$a^p \equiv a \bmod p$$

Induktionsschritt:

Mithilfe der Binomialkoeffizienten haben wir gezeigt, dass gilt: (siehe Sektion 4.7):

$$(a+1)^p = a^p + \left(\begin{array}{c} p \\ 1 \end{array} \right) a^{p-1} + \ldots + \left(\begin{array}{c} p \\ p-1 \end{array} \right) a + 1$$

Wir nehmen das ganze modulo p. Da alle Binomialkoeffizienten durch p teilbar sind, sind sie kongruent 0 bezüglich der Restklasse p. Folglich setze ich alle Summanden (äußer den beiden Randelementen) gleich Null.

$$\begin{aligned} (a+1)^p &\equiv a^p + 1 \bmod p \\ (a+1)^p - (a+1) &\equiv a^p + 1 - (a+1) \bmod p \\ (a+1)^p - (a+1) &\equiv a^p - a \bmod p \end{aligned}$$

Wir haben damit bewiesen, dass die Rechnung mit a das selbe ergibt, wie mit $a+1$. Der Induktionsschritt ist erfüllt. Der Beweis ist für alle natürlichen Zahlen erbracht. Eine Frage bleibt aber offen. Wieso habe ich vorhin den Term nicht vereinfacht? Dadurch wäre der Induktionsschritt ersichtlicher.

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

Der Grund ist, dass bei diesem Satz eine Bedingung erfüllt werden muss: a darf kein Vielfaches von p sein (löse die Gleichung einfach mit $a = k \cdot p$ und es wird sich ein

Widerspruch ergeben). Da auch gilt: p darf kein Vielfaches von a sein (sonst wäre p ja keine Primzahl), können wir auch sagen $\text{ggT}(a, b) = 1$.

$$a \neq k \cdot p$$
$$p \neq k \cdot a$$
$$\Rightarrow \text{ggT}(a, p) = 1$$

Wir zeigen noch ein Zahlenbeispiel:

$$(6+1)^{13-1} \equiv 6^{13-1} \bmod 13$$
$$1 \equiv 1 \bmod 2$$

Komischerweise ergibt das immer diese Eins; unabhängig davon welche Werte wir verwenden (solange sie der Definition entsprechen). Aber das ist auch das, was wir voraussetzten (den kleinen Fermat). Zusammengefasst gilt es:

Es sei $a \in \mathbb{Z}$ und $p \in \mathbb{P}$. Dann gilt:

$$a^p \equiv a \bmod p$$

Wenn zusätzlich noch $\text{ggT}(a, p) = 1$, dann gilt:

$$a^{p-1} \equiv 1 \bmod p$$

Wir rechnen noch ein paar Beispiele:

$$5^7 \equiv 5 \bmod 7$$
$$10^1 1 \equiv 10 \bmod 11$$
$$41317940^{6991} \equiv 41317940 \bmod 6991$$
$$41317940^{6991} = 1130 \bmod 6991$$

Hoppla... was ist beim letzten Beispiel passiert? Die beiden Seiten sind ja unterschiedlich. Falsch... die Sätze sind ident. Man muss bedenken, dass bei $a > p$ gilt, dass a modul(iert) wird. Sowohl 41317940 als auch 1130 entstammen der selben Restklasse ($\mathbb{Z}/6991\mathbb{Z}$) (Menge von Zahlen, die bei Division durch p bzw. m den selben Rest zurücklassen).

$$41317940 \bmod 6991 = 1130$$

```
#!/usr/bin/env python

primes = (3, 5, 7)

def fermat(a, p):
    # pow(a, b, c) == a**b % c
    return pow(a, p, p) == (a % p)

for number in xrange(1, 10000):
    for prime in primes:
        if not fermat(number, prime):
            print number, prime
            print 'Math Error'
```

4.16 Satz von Euler

Der Satz von Euler-Fermat lautet:

$$a^{\varphi(n)} \equiv 1 \bmod n$$

Der Großteil wurde bereits in der Sektion 4.15 genannt. Dort haben wir die Bedingungen aufgestellt, dass aus $a \in \mathbb{Z}$, $p \in \mathbb{P}$ sowie $\mathrm{ggT}(a, p) = 1$ folgt...

$$a^{p-1} \equiv 1 \bmod p$$

Wir haben bereits festgestellt, dass für eine Primzahl p die Eulersche Funktion $p - 1$ zurückgibt (siehe Sektion 4.11.2). Und das Faszinierende bei dem Satz von Euler: p muss gar keine Primzahl sein. Es seien $a, n \in \mathbb{Z}$ und φ die Eulersche Funktion.

$$a^{\varphi(n)} \equiv 1 \bmod n$$

Der Beweis kann über die Multiplikation folgen. Wenn für x und y gilt...

$$ax \equiv ay \bmod n$$

...wobei $\mathrm{ggT}(a, n) = 1$ dann gilt...

$$x \equiv y \bmod n$$

Wir können also eine Menge definieren für die gilt

$$r_1 \cdot \ldots \cdot r_{\varphi(n)} = r_1 \cdot \ldots \cdot r_{\varphi(n)} \cdot a^{\varphi(n)} \bmod n$$

Wir können nun die linke Seite durch die rechte Seite dividieren. Folglich erhalten wir den Satz von Euler:

$$a^{\varphi(n)} \equiv 1 \bmod n \qquad a, n \in \mathbb{Z}$$

Die genauen Eigenschaften der oben genannten Menge konnte ich jedoch nicht erfassen und sind deshalb nicht in diesem Dokument enthalten. Dies ist somit kein sauberer Beweis.

4.17 RSA – Rivest Shamir Adleman

4.17.1 Das Trio und die Entstehung

Die Kryptologen von RSA waren die drei Herren Ronald L. Rivest, Adi Shamir und Leonard Adleman am MIT ("Massachusetts Institute of Technology"), die 1977 eine Realisierung für Diffie und Hellmans Ideen fanden. Das Trio arbeitete gemeinsam im achten Stock in der Abteilung für Computerwissenschaften und Ron machte die beiden anderen auf die Standforder Diffie und Hellman aufmerksam. Nach ein bisschen Überzeugungsarbeit machten sich die Drei an die Arbeit. Rivest, ein genialer Computerwissenschaftler, entwickelte neue Theorien auf der Basis der zahlreichen von ihm gelesenen Fachartikel. Shamir konnte an den Kern jedes Kryptosystems dringen und erkennt Fehler unweigerlich. Adleman hatte die Aufgabe die Ergebnisse der beiden anderen zu überprüfen.

Eines Abends verbrachte das Trio die Nacht im Haus eines Studenten. Weil Rivest nicht schlafen konnte, holte er sich ein Mathematikbuch und las über Zahlentheorie. Als er über das Problem nochmals zu denken begann, kam ihm die Lösung der üblichen Verdächtigen: der Primzahlen. Er setzte sich an den Schreibtisch und verbrachte die ganze Nacht damit ein Dokument zu verfassen, das seine Ideen festhielt. Am Ende der Arbeit setzte er die Namen in alphabetischer Reihenfolge der drei Beteiligten drunter mit denen er schon seit langem an dem Rätsel kämpfte: Adleman, Rivest, Shamir.

Am nächsten Morgen kam dann die übliche Prozedur: Der Kryptograph verteilt das Werk und alle Kryptoanalytiker versuchen Fehler zu finden. Doch Adleman fand keinen Fehler. Mathematisch wäre das Verfahren nur über das Faktorisierungproblem lösbar. Die Arbeit war quasi perfekt, aber Adleman war mit etwas nicht ganz einverstanden. Das Dokument wäre nicht sein Werk und rein der Verdienst der beiden anderen. Nach einem Streit und einer Nacht später entschieden sich die drei Adlemans Namen an die letzte, unbedeutende Stelle zu setzen: RSA.

4.18 RSA-Mathematik

Jetzt können wir all unsere Überlegungen aus den vorhergehenden Kapiteln zusammenführen und die mathematische Korrektheit von RSA belegen.

Der kleine Satz von Fermat[6] besagt für $a \in \mathbb{R}$ und $p \in \mathbb{P}$ mit $\text{ggT}(a, p) = 1$ (die letzte Bedingung ist gegeben, da wir sagen müssen, dass a immer kleiner $p-1$ sein **muss**):

$$a^{p-1} \equiv 1 \bmod p$$

bzw.

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

Wir halten fest, dass der linke Term durch p mit einem Rest dividierbar ist. Eine der Rechenregeln aus der modularen Arithmetik[7] besagt...

$$a \equiv b \bmod m$$
$$\Rightarrow a^n \equiv b^n \bmod m$$

Daraus folgt:

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

Ist eine Zahl a durch n teilbar und b ebenfalls, so ist deren Produkt durch a und b teilbar. Das ist ein fundamentaler Satz der Teilbarkeit. Ich möchte es kurz an einem Beispiel erläutern: 2 teilt 6 und 3 teilt 6. Also teilt 6 auch das Produkt der beiden Zahlen 2·3 (der senkrechte Strich bedeutet "der linkte Term lässt sich durch den rechten Term ganzzahlig teilen").

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

In Restklassen rotieren Zahlen. Das bedeutet...

$$0 \equiv 0 \bmod a$$

$$a \equiv 0 \bmod a$$

$$k{\cdot}a \equiv 0 \bmod a$$

Das bedeutet jedes k-fache vom Argument a ist kongruent zu Null bezüglich des Moduls a.

$$a^{k(p-1)(q-1)} \equiv 1 \bmod p{\cdot}q$$
$$a^{1+k(p-1)(q-1)} \equiv a \bmod p{\cdot}q$$

Das lassen wir einmal so stehen und schauen uns die Kongruenzbedingung[8] an. Für jene gilt:

$$e{\cdot}d \equiv 1 \bmod \varphi(N)$$

[6] Sektion 4.15
[7] Sektion 4.8.3
[8] Sektion 4.18.1

$$\Rightarrow e{\cdot}d = 1 + k{\cdot}\varphi(N) \bmod \varphi(N)$$
$$\Rightarrow e{\cdot}d = 1 + k{\cdot}(p-1)(q-1) \bmod \varphi(N)$$

Und jetzt haben wir alle Sätze fertig gestellt. Es gilt für die Verschlüsselung des Klartext m die Formel und das Gesamte wird mit d potenziert (da zuerst Verschlüsselung und dann Entschlüsselung). In Gleichungen ausgedrückt:

$$(m^e)^d \equiv m^{ed} \equiv m^{1+k(p-1)(q-1)} \equiv m^1 \equiv m \bmod p{\cdot}q$$

4.18.1 Ein Beispiel mit RSA

In diesem Kapitel werden die cryptotools verwendet; ein beiliegendes Paket von mir programmiert in python[9]

Wir wählen zwei Primzahlen p und q. Während sichere Programme Primzahlen im Bereich 2^{2048} berechnen, verwenden wir die Primzahlen 7 und 17, um es an einem einfachen Beispiel zu verdeutlichen. Wir definieren N als das Produkt beider Primzahlen.

$$N = p * q \qquad\qquad 119 = 7 * 17$$

Für 7 und 17 gilt $\varphi(N) = \varphi(p{\cdot}q) = (p-1)(q-1) = 96$. Wer sich noch nicht davon überzeugen lässt, dass $\varphi(p{\cdot}q) = (p-1)(q-1)$ gilt (siehe Sektion 4.11.2), kann die Zahlen in die cryptotools eingeben, die diesen Trick nicht ausnutzen (dazu wäre nämlich ein Primzahltest notwendig, der einen Geschwindigkeitsverlust bedeutet).

```
Which option would you like to start? c
number: 119
96
```

Argument	Wert
p	7
q	17
N	119
$\varphi(N)$	96

Jetzt benötigen wir eine beliebige – zu $\varphi(N)$ teilerfremde – Zahl e, die kleiner als 72, aber größer als 1 ist.

[9] http://lukas-prokop.at/proj/spezialgebiete/tools.tar.gz

Which option would you like to start? z
number: 96
71

Es gilt $e = 71$. Bezüglich e benötigen wir ein multiplikativ modular inverses Element zu e. In den reelen Zahlen ist ein multiplikativ inverses Element eine Zahl b, die diese Gleichung erfüllt...

$$a{\cdot}b = 1$$

Dabei handelt es sich immer um $\frac{1}{a}$. Wenn wir von modular multiplikativ inversen Elementen sprechen, muss es eine Zahl d geben, die diese Gleichung erfüllt...

$$e{\cdot}d \equiv 1 \bmod \varphi(N)$$

Und diese Aufgabenstellung kennen wir aus Sektion 4.14.

$$96 = 1{\cdot}71 + 25$$
$$71 = 2{\cdot}25 + 21$$
$$25 = 1{\cdot}21 + 4$$
$$21 = 5{\cdot}4 + 1$$
$$4 = 4{\cdot}1 + 0$$
$$\mathrm{ggT}(96, 71) = 1$$

Wieso ist der ggT Eins? Weil wir oben festgelegt haben, dass die beiden teilerfremd sein müssen.

$$1 = 21 - 5{\cdot}4$$
$$1 = 21 - 5{\cdot}(25 - 1{\cdot}21) = 6{\cdot}21 - 5{\cdot}25$$
$$1 = -5{\cdot}25 + 6{\cdot}(71 - 2{\cdot}25) = 6{\cdot}71 - 17{\cdot}25$$
$$1 = 6{\cdot}71 - 17{\cdot}(96 - 1{\cdot}71) = 23{\cdot}71 - 17{\cdot}96$$
$$1 \equiv 23{\cdot}71 - 17{\cdot}96 \bmod 96$$
$$1 \equiv 23{\cdot}71 \bmod 96$$
$$1 \equiv d{\cdot}e \bmod \varphi(N)$$

Wie wir sehen ist die -17 unbedeutend und wird weggeworfen. Nach dieser Schlüsselerzeugung haben wir alle Parameter, um Nachrichten mittels RSA zu verschlüsseln.

Argument	Wert	Status
p	7	löschen
q	17	löschen
N	119	öffentlich
$\varphi(N)$	96	löschen
e	71	öffentlich
d	23	privat
k	-17	löschen

Wir möchten nun eine Nachricht verschlüsseln. Eine lange Zahl würde eine Vielzahl von Berechnungen erfordern, deshalb nehmen wir nur einen Buchstaben. Gemäß ASCII[10] wird der Buchstabe R als 82 kodiert.

$$C = M^e \bmod N$$
$$C = 82^{71} \bmod 108$$

Wir nutzen SaM um die Gleichung zu lösen (Sektion 4.9.2)

> Which option would you like to start? B
> decimal number: 23
> 10111

Das bedeutet wir rechnen "QQMQMQM". Also

$$C = ((((108^2)^2 \cdot 108)^2) \cdot 108)^2 \cdot 108 \bmod 119$$
$$C = 108$$

Wir haben die 82 als 108 verschlüsselt. Nun möchten wir entschlüsseln.

$$M = C^d \bmod N$$
$$M = 108^{23} \bmod 119$$
$$M = 82$$

Wir bezeichnen N als RSA-Modul, e als Verschlüsselungs- und d als Entschlüsselungskoeffizienten. Wir erhalten wieder 83 (=S) und haben somit gezeigt, dass eine Verschlüsselung mit RSA möglich ist. Dabei muss der Kommunikationspartner weder verfügbar sein (wie beim Diffie-Hellman-Schlüsselaustausch) noch ist ein Man-in-the-middle möglich. Der private Schlüssel ist das Geheimnis auf dem die Sicherheit des Verfahrens beruht und wir sehen auch: Kerckhoffs wäre stolz auf die drei Herren gewesen.

[10] American Standard Code for Information Interchange

4.18.2 Die RSA-Problematik

> Ich sage dir die Zahl 6 und du machst eine Primfaktorzerlegung

Ok... das war nicht schwer. $2 \cdot 3 = 6$.

> Ich sage dir die Zahl
> 114381625757888867669235779976146612010218 2
> 96721242362562561842935706935245733897830 59
> 71235639587050589890751475992900268795435 41
> und du machst eine Primfaktorzerlegung

Plötzlich erscheint das Problem in anderen Dimensionen. Bislang sind keinerlei Verfahren bekannt, die dieses Problem effizient berechnen können. Ein analoges Beispiel wäre es ein Ei zu zerschlagen. Ein Ei kann man sehr leicht kaputt machen, aber es wieder zusammenzufügen, scheint unmöglich zu sein. Oder man mischt Sand mit Zucker. Wenn man Sand und Zucker trennen möchte, braucht man sehr viel Zeit.

Im Falle dieser 129-stelligen Zahl benötigt man 45 Stunden. 600 Freiwillige haben 1994 8 Monate lang Kongruenzen gesucht und an einen Supercomputer gesandt. Der brauchte dann diese 45 Stunden, um die Zahl zusammenzusetzen. War doch klar, dass die Zahl das Produkt dieser beiden Zahlen ist, oder?

> 3490529510847650949147849619903898133417764 63
> 8493387843990820577 * 32769132993266709549961
> 9881908344614131776429679929425397982885 33

Es ist übrigens so, dass Verschlüsselungsalgorithmen heute das "30-Jahre-Kritierium" erfüllen müssen. Bei dieser Problematik liegt es nur am Problem der Geschwindigkeit. Kann man die Primfaktorzerlegung schneller durchführen, dann kann die Zahl geknackt werden. Und wenn wir uns die letzten Jahre anschauen, so werden Computer immer schneller. Es ist jetzt Ziel jedes Algorithmus', mindestens die nächsten 30 Jahren den Entwicklungen standzuhalten.

4.18.3 Attacke auf RSA

Fixpunkte

Ein Grundproblem hat RSA. Es wurde nicht dafür konzipiert Fixpunkte auszugleichen. Als ich meine ersten Beispiele mit eigenen Koeffizienten rechnete, landete ich mehrmals bei Fixpunkten. Anscheinend ist bei niedrigen Primzahlen die Gefahr höher Fixpunkte zu bekommen. Egal... es spielt eigentlich keine Rolle und die Sicherheit von RSA wird

nicht gefährdet. Allerdings passierte es mir einmal, dass sich $82^3 \bmod 8 = 0$ ergibt. In dem Fall gilt jedoch, dass der zu verschlüsselnde Buchstabe kleiner sein muss als das RSA-Modulo. In diesem Beispiel ist das nicht der Fall und dadurch ist wohl der Fehler aufgetreten. Die Fixpunkte in RSA haben keinerlei Auswirkung auf die Funktionsweise oder Funktionalität des Verfahrens.

Koeffizienten und Module

N	öffentlich
e	öffentlich
d	privat

Welchen Koeffizienten benötigen wir zur Entschlüsselung? d. Das bedeutet wir müssen einen Angriff starten, um d errechnen zu können. Woraus ist d entstanden? Es ist das modular inverse Element von e, wobei uns e bekannt ist. Das heißt wir können mit dem erweiterten euklidischen Algorithmus ganz normal d berechnen...

$$e{\cdot}d \equiv 1 \bmod \varphi(N)$$

...wenn wir $\varphi(N)$ hätten. Und damit wird klar, dass $\varphi(N)$ die wirklich geheime Zahl ist, die wir erhalten wollen. Wer $\varphi(N)$ hat, hat alle Koeffizienten. Besitzen wir e (da öffentlich), N (da öffentlich) und $\varphi(N)$ (als worst-case Szenario) so können wir d mit dem erweiterten euklidischen Algorithmus berechnen, k benötigen wir in keinem Fall und die beiden Primzahlen können wir dann auch leicht aus einem Gleichungssystem erhalten.

$$\varphi(N) = 24624$$
$$N = 24961$$

$$(p-1)(q-1) = 24624$$
$$p{\cdot}q = 24961$$

$$p{\cdot}q - q - p + 1 = 24624$$
$$24961 - q - p = 24623$$
$$-q - p = -338$$
$$q + p = 338$$

Durch ganz einfaches Herumprobieren (p und q müssen Primzahlen sein) kommt man sehr schnell an die Lösung. Haben wir jetzt auch p und q berechnet, so haben wir alle Koeffizienten berechnet; nur durch die Kenntnis von $\varphi(N)$.

Aber ist es nicht möglich von N auf $\varphi(N)$ zu schließen. Wir kennen doch die Definition von $\varphi(x)$: Gibt die Anzahl der teilerfremden Zahlen kleiner x zu x zurück. Das bedeutet wir müssten zur Errechnung von $\varphi(x)$ von jeder Zahl kleiner x den größten gemeinsamen Teiler von der Zahl mit x überprüfen. Ist der ggT Eins, so ist sie teilerfremd zu x und ein Zähler wird inkrementiert. In der Tat geht dies sehr schnell. Der euklidische Algorithmus zur Berechnung von ggTs wurde bereits vorgestellt (Sektion 4.11.1). Allerdings muss man bedenken, dass die Berechnung für jede Zahl geschehen muss.

```
num = 540
zaehler = 0

for i in xrange(num):
    if ggT(i, num) == 1:
        zaehler += 1

print zaehler
```

Die for-Schleife lässt bereits Schlechtes vorahnen. Bei einer Zahl wie $num = 540$ halten sich die Berechnungen in Grenzen, aber bei riesigen Zahlen (wie wir wissen verwendet man bevorzugt 617-stellige 2048-Bit-Zahlen) ist das ein enormer Geschwindigkeitsverlust. $\varphi(N)$ lässt sich aus N **nicht** effizient berechnen. Wie berechnet es aber die Person selbst? Sie ist ja im Besitz von p und q und genau dort ist der Knackpunkt. $\varphi(N)$ lässt sich ineffizient über den Euklidischen Algorithmus berechnen oder effizient durch eine einfache Multiplikation der Form $(p-1)(q-1)$. Und das ist auch die Stelle, wo klar wird, was mit Falltürfunktion gemeint ist. Das Produkt von p und q lässt sich ganz leicht berechnen. Ebenso lässt sich das Produkt von $(p-1)$ und $(q-1)$ leicht berechnen. Aber $\varphi(N)$ lässt sich nicht leicht berechnen; außer man ist im Besitz von p und q. Und diese Primzahlen p und q besitzt nur der Kodierer selbst. Und genau das ist der Knackpunkt. Wer N in p und q zerlegt kann und dann daraus $(p-1)$ und $(q-1)$ berechnet, kann $\varphi(N)$ berechnen und damit einen RSA-Schlüssel knacken. RSA ist gebrochen, wenn jemand N effizient faktorisieren kann.

Eine klassische Attacke

Effiziente Faktorisierungsverfahren gibt es nicht. Quadratisches Sieb und die Probedivision sind die beiden führenden Verfahren, die eingesetzt werden. Die werden auf verteilten Rechner ausgeführt und so wird der riesige Aufwand auf viele kleiner Computer (Stichwort Verteiltes Rechnen) aufgespalten. Wie sieht diese Vorgehensweise aus?

- Die Probedivision ermittelt kleine mögliche Faktoren

- Durch einen Primzahltest wird ermittelt, ob es sich um eine Primzahl oder Primpotenz handelt
- Mithilfe der elliptischen Kurven wir nach kleinen Primfaktoren gesucht ($< 10^30$)
- Mit dem Quadratischen Sieb (für Zahlen mit weniger als 120 Dezimalstellen) oder dem Zahlkörpersieb wird faktorisiert.

4.19 Anwendung von RSA für den Benutzer – PGP und GPG

Whitfield (Abschnitt 4.3) hatte Recht. Er sah das Informationszeitalter voraus und tatsächlich gab es ein verstärktes Aufkommen von E-mailtransfer und Datenaustausch (abseits von Spam). Bevor wir aber beginnen Verfahren zu entwickeln, die uns Onlineabstimmungen ermöglichen, müssen wir uns unserer Privatsphäre bewusst werden und mit Verschlüsselung sorgfältig verwenden, damit wir nicht 25 Jahre[11] rückwärts gehen. Und wie kann man das RSA-Verfahren besser testen als wenn es Einzug in unseren Alltag erhält?

Phil Zimmermann war in den 70ern ein Physik- und Computerwissenschaften-Student in Florida. Doch statt sich auf seine zwei Fächer zu konzentrieren, beschäftigte ihn vor allem die aktuelle politische Situation. Die Sowjetunion (mit Breschnew) und Ronald Reagan machten ihm solche Sorgen, dass er beschloß den Kampf von daheim aus zu beginnen. Er zog doch nicht nach Neuseeland und förderte die nukleare Abrüstung Amerikas. Bei einer Demonstration wurde er als einer von mehr als 400 Demonstranten verhaftet. Nach seiner Freilassung wurde ihm, dass jede politische Organisation im Visier der Behörden steht und überwacht werden kann bzw. überwacht wird. Er wurde auf die kryptographischen Entwicklungen rund um RSA aufmerksam und setzte sich das Ziel kryptographische Techniken für Bürger zugänglich zu machen.

> But with the coming of the information age, starting with the invention of the telephone, all that has changed. Now most of our conversations are conducted electronically. This allows our most intimate conversations to be exposed without our knowledge. Cellular phone calls may be monitored by anyone with a radio. Electronical mail, sent across the Internet, is no more secure than cellular phone calls. Email is rapidly replacing postal mail, becoming the norm for everyon, not the novelty it was in the past. [...]
>
> Advances in technology will not permit the maintenance of the status quo, as far as privacy is concerned. The status quo is unstable. If we do nothing, new technologies will give the government new automatic surveillance capabilities that Stalin could never have dreamed of. The only way to hold the line on privacy in the information age is strong cryptography.[10]

[11] 1984: kritischer Roman zum Stichwort Überwachungsstaat von George Orwell

> If privacy is outlawd, only outlaws will have privacy. Intelligence agencies have access to good cryptographic technology. So do the big arms and drug traffickers. So do defence contractors, oil companies, and other corporate giants. But ordinary people and local and laternative political organisations mostly have not had access to affordable ways of protect their privacy [...] Privacy is a right like any other. You have to exercise it or risk losing it.

Die Veränderung ist klar ersichtlich: Wenn jemand einen Brief am Postamt aufgibt, so hat er genau im Überblick wem er die Post übergibt und bezahlt denjenigen auch dafür. Treten Manipulationen auf, so bemerkt das spätestens der Empfänger. Wenn jemand jedoch eine E-mail schreibt, so handelt es sich um digitale Daten, die leicht auswertbar sind. Man übergibt die E-mail seinem Server und die E-mail begibt sich auf eine Reise. Ein Ausleseprozess (oder gar eine Manipulation) würde keine Spuren hinterlassen. Der Empfänger würde nichts bemerken.

Wenn man Kryptographie der Allgemeinheit zugänglich macht, so gibt man jedem Bürger das Recht auf seine eigene Privatsphäre. Niemand kann Inhalte lesen (außer jeder Person, die RSA gebrochen hat) und auswerten. Das betrifft sowohl die Terroristen wie auch den Staat, der seine politische Stabilität schützen muss. Phil Zimmermann war derjenige, der "Pretty Good Privacy" (PGP; eine RSA-Implementierung) schrieb und es ins Internet stellte. Laut den USA war dies ein Verstoß gegen die Exportbeschränkungen für Waffen, da kryptographische Verfahren dem Waffengesetz untergestellt sind und PGP nicht exportiert werden darf. Der Hintergrund war die Frage, ob die USA Verfahren zulassen sollten, die auch die NSA nicht mehr knacken kann und damit Bürger nicht mehr überwachen kann.

Wir müssen noch ein Problem an RSA ansprechen. RSA ist langsam. Furchtbar langsam um genau zu sein und RSA kann nicht mit anderen Verfahren (wie DES[12]) mithalten. Deshalb wurden *hybride Verfahren* angewandt. Der Schlüssel wird mit RSA übertragen, aber die Nachricht selbst wird mit einem symmetrischen Verfahren (Phil Zimmermann verwendete IDEA, welches DES ähnelt) verschlüsselt. Das hybride Verfahren löst also das Problem des Schlüsseltausches durch asymmetrische Verfahren und nutzt den Geschwindigkeitsvorteil der symmetrischen Verfahren.

Phil Zimmermann gewann den Prozess. 1996 durfte er ganz frei sein Produkt verbreiten und die amerikanische Behörde konnte nicht mehr mitreden. Aber wesentlichen Einfluß hatten die Ereignisse während des dreijährigen Prozesses. Denn er schrieb den Quelltext der PGP-Software in einem Buch nieder. Dadurch war die Software nach amerikanischem Recht kein kryptographisches Werkzeug mehr, sondern ein Buch welches exportiert werden durften. Angelangt in Europa tippten zahlreiche Freiwillige den Quelltext ab, kompilierten das Programm und verbreiteten es frei. Phil Zimmermann erreichte ein Vielzahl von E-mails (großteils von Menschenrechtsorganisationen), die sich bei Phil Zimmermann bedankten und nun mit einem Sicherheitsgefühl kommunizieren konnten. Natürlich waren die E-mails verschlüsselt.

[12] Data Encryption Standard

Mit GPG ("GNU Privacy Guard") steht jedem UNIX-Benutzer eine vollständige, funktionstüchtige Umgebung mit RSA zur Verfügung.

4.20 Authentifikation und Integrität

Was Rivest, Shamir und Rivest nicht bedachten, war die digitale Signatur. Also ein Verfahren welches uns erlaubt zu verifizieren, dass unser Partner wirklich unser Partner ist und nicht bloß ein Hacker, der deine Nachrichten sammelt, um daraus vielleicht Inforamtionen zu deinem privaten Schlüssel extrahieren zu können. Das ist recht einfach; wir müssen das Verfahren bloß umkehren.

Bisher verwendeten wir den öffentlichen Schlüssel (des Partners) um eine Nachricht verschlüsseln zu können und den privaten Schlüssel um

4.21 Hashes

Unter Hashes versteht man den Code einer Einwegfunktion. Die Hashes zielen damit nicht auf die Bestrebungen ab, die wir bisher verfolgten: Das Verschlüsseln einer Nachricht von Person A und Entschlüsseln der Nachricht von Person B. Nun geht es uns gar nicht um den Inhalt der Nachricht. Sagen wir, wir hätten zwei Eingaben zu zwei verschiedenen Zeitpunkten. Wir möchten nun testen ob die erste Eingabe der zweiten Eingabe entspricht. Wir möchten es jedoch vermeiden den Inhalt im Klartext zu speichern, da er hochsensibel ist. Also sollte es auch nicht möglich sein aus dem gespeicherten Code den Klartext zu extrahieren.

Als Erstes möchten wir die Frage diskutieren, wie der Begriff der Kodierung hier verwendet werden muss. Sollte ein Element der Menge A einem, zwei oder gar keinem Element der Menge B zugeordnet werden?

$P + S < C$
Wir könnten nun argumentieren, dass es durchaus sinnvoll wäre einen Klartext in mehrere Codes resultieren zu lassen. So könnte das Wort "Krypto" sowohl dem Buchstaben A wie auch B zugeordnet werden ($C = (AvB) = f(M)$). In der Praxis ist solch ein Algorithmus absolut unbrauchbar. Wenn ich mich in einem Benutzersystem anmelde, erwarte ich eine rasche und unkomplizierte Antwort. Bei der Frage $B == f(M)$ muss die Antwort sofort klar sein und darf nicht vom Zufall abhängen. Vom Zufall hängt es jedoch ab, wenn ein Klartext einmal in Code XY und einmal in Code YZ resultiert.

$P + S > C$
Ist die Anzahl an Codestücken kleiner als die Anzahl der Eingaben, besteht Kollisionsgefahr. Unter Kollision bezeichnet man das Ereignis, wenn zwei unterschiedliche Eingaben zur gleichen Codesequenz führen. Dies hätte ebenfalls eine gravierenden Fehler in sich, da die Anzahl an Möglichkeiten sinkt und dadurch die Sicherheit des Algorithmus verringert

wird.

$P + S == C$
Wir haben gesehen, dass es weder zielführend ist zu viel Code bzw. zu wenig Codesequenzen zu besitzen. Alle modernen Hashfunktionen erzeugen einzigartige Codes und erfüllen damit den Begriff der Kodierung.

4.22 Fingerprint

Ich möchte kurz den Begriff des Fingerprints ansprechen. Wenn man sich auf Key-Signing-Parties trifft, ist es mühsam sich mit einem 3-seitigen Ausdruck zu treffen und zu überprüfen, ob der akzeptierte Schlüssel jener deines Freundes ist. Wenn man vor eine solchen Problematik stehen würde, nimmt man wohl eher Stichproben heraus. Man überprüft die ersten 10 Zeichen, danach die ersten 10 Zeichen auf der nächsten Seite, entdeckt dass der Freund eine andere Schriftgröße ausgedruckt hat und damit alles auf dem Papier verschoben ist und man ärgert sich. Viel einfacher ist es hier ein System zu haben, welches den Schlüssel wesentlich kürzt, aber trotzdem noch eindeutig ist. Bei diesem Punkt kommen Fingerprints ins Spiel. Fingerprints sind Hashes von Schlüsseln. Ich besitze einen Schlüssel, erzeuge einen kurzen Hash davon und gehe zu meinem Freund. Dieser erzeugt ebenfalls einen Hash aus meinem Schlüssel. Wir überprüfen unsere beiden kurzen Hashes und wir erkennen daran, ob ein Man-in-the-middle unsere öffentlichen Schlüssel manipulierte.

Wir müssen uns noch einer Sachen bewusst werden. Unser Dezimalsystem und die 30 deutschen Buchstaben sind Konzepte mit denen unsere Rechner schwer umgehen können. Computer haben Binärwerte und ASCII lieber. Wir sind trotzdem nicht auf die leichter berechenbaren Konzepte umgestiegen (und verwenden es heute nicht in unserer Alltagssprache), da der Mensch ein Problem hat sich Stellen zu merken. Wenn wir uns CRY zu merken versuchen, wird es uns viel leichter fallen als bei 010000110101001001011001 (CRY = ASCII 67 82 89 mit 8-Bit-Darstellung). Unser Gehirn merkt sich einfach viel leichter ein komplexes Alphabet – welches es regelmäßig anwendet – mit wenig Stellen als eine vielstellige Zahl mit einem Minimalalphabet.

Jetzt verstehen wir auch wieso wir niemals Nullen und Einsen zu sehen bekommen, wenn wir nach Schlüsseln googeln. Die Bits werden so umgeschrieben, dass wir sie als ASCII-Zeichen empfangen. Die Nachricht wird dadurch bereits wesentlich kürzer. Wenn wir nun ASCII mit Sonderzeichen erweitern und die Breite von Unicode ausnutzen, dann können wir eine anfangs millionen-stellige Zahl auf ein paar Sonderzeichen reduzieren. Diese beiden Hashes zu vergleichen, ist natürlich wesentlich leichter. Ein PGP-Schlüssel mit Sonderzeichen besteht nur aus ca. 1756 Zeichen.

Wir müssen uns jedoch etwas eingestehen. In der Realität ist die Sicherheit bei Hashfunktionen, die zum Fingerprinting genutzt werden, niedrig. Man akzeptiert sehr viele Kollisionen und genießt dafür den Komfort, dass man kürzere Hashes erhält. Allgemein

kann man eigentlich von einer eigenen Kategorie von Hashes sprechen: Die Anzahl der Eingabewerte übersteigt die der Ausgabewerte erheblich. Eine Kollision ist unvermeidbar. Dies passiert teilweise unabsichtlich. Der User gibt zB einen Usernamen bekannt, der nicht hervorgesehene Zeichen enthält. Was soll der Algorithmus machen? Fehler werfen, Zeichen ignorieren oder trotzdem kodieren?

Im Bereich wo es absichtlich gemacht wird, bildet sich wieder eine neue Disziplin: Finde einen Algorithmus, der die Kollisionen möglichst gleichmäßig auf alle Ausgaben verteilt.

4.23 RSA-Community – wer und wie RSA verwendet wird

Aber man kann die Frage nach sicherer Kommunikation noch ein Stück weiter treiben: Ich weiß zwar, dass jenes Gegenüber schon eine Weile mit mir kommuniziert und niemand hat zwischendurch die Leitung gestört. Aber ist mein Gegenüber trotzdem die Person, die ich vor einem halben Jahr auf einer Party kennen lernte? Die Frage ist nicht in dieser Form beantwortbar. Man *muss* sich treffen, um zu beweisen, dass der öffentliche Schlüssel nur dir zuordenbar ist.

Auf Informatikertreffen finden oft sogenannte KeySigningParties statt. Jeder Benutzer nimmt dazu einen Lichtbildausweis und seinen Fingerprint (Hash seines öffentlichen Schlüssels) mit. Computer dürfen auf keinen Fall mitgenommen werden, da sonst Passwörter ausspioniert werden könnten. Im Laufe der Party überprüfen die Teilnehmer, ob ihre aktuell verwendeten Schlüssel mit den Fingerprints der Besitzer überein stimmen. Typischerweise wird eine E-Mail geschickt, die kontrollieren soll, ob die angegebende E-mailadresse (am Keyserver) wirklich zur dieser Person gehört. Auf diese Weise wird der höchste Grad an Sicherheit gefordert. Das so entstehende "Web of Trust" ermöglicht einen dezentralisierten Umgang mit Schlüsseln. Es gibt keine zentrale Stelle, die alle Schlüssel verifiziert. Sondern ich vertraue jedem, dem jemand vertraut, dem wiederum ich selbst vertraue. Wenn also Bob Alice vertraut und Alice Eve, dann vertraut Bob Eve.

Kapitel 5

Zukunft

> Menschliche Erfindungskraft kann kein Verschlüsselungssystem schaffen, das durch menschliche Erfindungskraft nicht gebrochen werden könnte. – Edgar Allen Poe

5.1 Zusammenfassung der Vergangenheit

Nachrichten zu verschleiern war immer im Interesse der Menschheit. Die Kryptologie gibt es so lange wie Menschen fähig sind, Information zu notieren. Die individuelle Handschrift kann jeder Mensch so verschnörkeln, dass er selbst – aber kein anderer – die Schrift lesen kann. Kommunikation ist in dem Moment nicht mehr möglich in dem das Gegenüber deine Sprache nicht mehr versteht. Eltern redeten gerne Französisch untereinander, weil sie nicht wollten, dass die Kinder mitreden können. Jeder Komponist hat ein paar neue Notenzeichen eingesetzt. Ein paar haben sich durchgesetzt. Leonardo da Vinci bevorzugte es Spiegelschrift zu schreiben. Hat man es sich einmal angewöhnt, fällt es immer leichter und andere Menschen können nur schwer den Text lesen. Wie wir sehen wurde Steganographie gerne eingesetzt.

Ab der monoalphabetischen Substitution können wir von einer Revolution der Kryptographie reden. Es war ein ständiger Kampf zwischen Kryptoanalytikern und Kryptographen Nachrichten zu ver- bzw. entschlüsseln. Es war ein ständiger Kampf das Werk des anderen zunichte zu machen. Es ging darum die Anzahl der möglichen Schlüssel zu erhöhen, damit die Laufzeit der Algorithmen unendlich wird und die einzig immer funktionierende Variante (Brute-Force) ineffizient wird.

Genauso können wir einen Wandel in den Methoden entdecken. Da anfangs keine komplexen Algorithmen verwendet wurden, konnte der Autor selbst die Nachricht verschlüsseln. Seit dem 1. Weltkrieg sind komplizierte Algorithmen bekannt, die der Autor ohne mathematische Kenntnisse nicht selbst durchführen kann. Meist wurden die kryptographischen Methoden von einer benutzerfreundlichen Oberfläche umgeben, um es dem Autor zu er-

leichtern. In den Kinderschuhen der Kryptographie waren es noch Linguisten, die die Entwicklung von Instrumenten übernahmen, aber seit dem 2. Weltkrieg sind es Informatiker, Mathematiker und andere Wissenschaftler. Mit Maschinen versucht man das De- bzw. Chiffrieren zu erleichtern. Dem Computerzeitalter sei es zu verdanken, dass hinter uns ständig Kodierungen ablaufen und wir komfortabel kaum etwas mitbekommen. Egal ob man beim Homebanking das Passwort eingibt oder SSL-Zertifikate deine Identität verifizieren oder dein Passwort auf einer Webseite als Hash in einer Datenbank landet oder dein Betriebsystem dein Passwort als Hash in der /etc/shadow salted und ablegt – du bekommst glücklicherweise recht wenig davon mit.

In dem Moment in dem die Benutzer darauf vertrauen, dass ihr Passwort sicher gespeichert wird, ist es zugleich die Aufgabe der Kryptographen sichere Verschlüsselungsalgorithmen zu schaffen. Es ist eine Verantwortung für die Sicherheit der Endbenutzer zu sorgen. Und damit ist zugleich ein wirtschaftliches Interesse vorhanden, weshalb Kryptographie weiterhin ein Bestandteil unseres Lebens bleiben wird.

Aktuell geht es um Algorithmeneffizienz und Schlüsselmöglichkeiten. Je schneller der Algorithmus, desto besser für die Kryptoanalytiker. Je größer die Anzahl der möglichen Schlüssel, desto besser für die Kryptographen. Allerdings wird es einen großen Bruch geben. Wir sind immer davon ausgegangen, dass wir einen Code haben auf den wir den Algorithmus unendlich mal anwenden können. Mit der Quantenkryptographie ändert sich das. Zwar freuen sich Kryptoanalytiker auf die schnelleren Quantenrechner, aber sie stehen vor einem anderen Problem: Ein Algorithmus (ein Filter) kann nur einmal angewandt werden. Ist er falsch, erhält der Kryptograph sofort Kenntnis darüber, dass der Code fehlerhaft entschlüsselt wurde.

Aber wer weiß: Auch Cäsar hielt seinen Code für unknackbar. Immer wieder konnten Kryptoanalytiker ihr Geschick unter Beweis stellen. Aber eine Gesellschaft wo man keine Furcht vor der Äußerung der eigenen Meinung haben muss: Die können wir gebrauchen und damit wird sie die Menschheit anstreben.

5.2 Laufzeitproblem

5.3 Quantentheorie

Ganz neue Möglichkeiten werden Quantencomputer eröffnen. Quantencomputer arbeiten mit Geschwindigkeiten, sodass "gegenwärtige Computer wie kaputte Rechenschieber dagegen aussehen"[6]. Damit wir die Ideen verstehen, müssen wir kurz die Quantenmechanik beleuchten. Wie immer treffen wir bei unserer Suche nach großen Wissenschaftern mit breit gefächertem Wissen auf Personen, die aus der Kryptologie kommen.

In unserem Fall ist das Thomas Young (* 1773 † 1829). Der Forscher war wesentlich daran beteiligt die Hieroglyphen zu entschlüsseln. Leider knackte die Entschlüsselung ein französischer Kollege, doch Thomas Young war mit seinen breiten Sprachkenntnissen

dicht auf den Fersen. Die Entschlüsselung der Hieroglyphen ist eines der spannendsten Bindeglieder zwischen Geschichte und Kryptologie.

Die Physik zu Zeiten Thomas Youngs hatte keine Erklärung, was denn Licht sei. Doch er begann zu experimentieren. Er maß die Wellenlänge von Licht und entdeckt, dass sie von der Farbe abhängig ist. Er führte sein berühmtes Doppelspaltexperiment durch. Durch 2 Spalten werden Lichtstrahlen gesandt und auf dem Bildschirm auf der anderen Seite entsteht ein Streifenmuster. Dieses Streifenmuster konnte man durch die Auffassung, dass Licht aus Wellen bestehe und Interferenzen auftreten, erklärt werden. Also Thomas Young jedoch mit einem schwachen Glühfaden einzelne Lichtwellen durchsandte, entstand dieses Muster aus. Ab dem Moment war die Physik ratlos und Licht musste sowohl als Teilchen als auch Welle aufgefasst werden. Wir nennen dieses Teilchen Lichtquantum oder einfacher: Photon. Wir wissen nie durch welchen Spalt es geflogen ist und was es in der Zwischenzeit angestellt hat.

Es gibt kein analoges Objekt in der klassischen Physik, welches die selben Eigenschaften aufweist. Das macht die Quantenforschung so kryptisch.

Quantenphysiker vertreten hier zwei Lager. Entweder spaltet sich das Universum in dem Moment in zwei Universen und das Photon fliegt in den beiden Universum durch unterschiedliche Spalten (Vielwelten-Deutung) oder das Photon fliegt durch beide Spalten gleichzeitig und steht in Wechselwirkung mit sich selbst (Superpositions-Prinzip).

David Deutsch war der erste Mensch, der die Konzepte der Quantentheorie auf den Computer (der mit klassischer Physik funktioniert) übertragen wollte. Er gilt als der Pionier der Quantencomputer.

Der Vorteil der Quantencomputer ist die oben angesprochene Überlagerung von Zuständen. Wenn man nicht weißt, was das Photon macht, kann es alles machen. Ein klassischer Computer bekommt zwei Fragen gestellt, sucht den passenden Algorithmus und arbeitet eine Frage nach der anderen ab. Durch Prozesse, Threads und Prozessorkerne versucht man software- und hardware-technisch Parallelität zu erzeugen, aber Quantencomputer arbeiten mit wirklicher Parallelität. Bei Quantencomputer können wir die Fragen als Überlagerung von Zuständen zusammgefasst eingegeben werden. Die Maschine würde dann in eine Superposition von Zuständen übergehen und für jede Frage entsteht ein Zustand. Durch diese Parallelität hat der Quantencomputer eine wesentlich höhere Leistung.

Wie arbeitet ein Quantencomputer? Prinzipiell wird auch im binären System gearbeitet. Photonen können sich in zwei Richtungen drehen: östlich oder westlich. Dabei kann jede Richtung einen binären Zustand bezeichnen. Wenn wir von binären Werten sprechen, sprechen wir von Bits; einer Folge von binären Buchstaben. Wenn wir von binären Werten bei Quantencomputern sprechen, nennen wir sie Quantenbits oder kurz Qubits.

Wer eine Überlagerung von zwei Zuständen zugleich schafft, schafft zwei Berechnungen zur selben Zeit. Wer eine Überlagerung von x Zuständen zugleich schafft, kann x Berechnungen zur selben Zeit ausführen. So hat man es geschafft mit einem Quantenrechner mit

einer Leistung von 7 Qubits die Zahl 15 in die Primzahlen 5 und 3 zu faktorisieren. Der verwendete Algorithmus wurde 1994 von Peter Shor entwickelt. Eine Shor-Algorithmus-Implementierung knackt einen RSA-Schlüssel mit 1024 Bits in Sekunden. Nur sieht die Implementierung einen Quantencomputer vor und die Wissenschaftler hatten damals keine Ahnung wie ein solcher zu konstruieren wäre. 1996 wurde dann ein Listenabsuch-Algorithmus von Lov Grover entwickelt. Mit ihm wäre es möglich DES zu knacken. Mit Quantencomputern wäre RSA und jedes bisherig bekanntes Verschlüsselungsverfahren geknackt, welches auf dem Faktorisierungsproblem beruht. Schlüssel wie 2048 Bit klingt heute viel, doch schon in ein paar Jahren könnten diese Schlüssel in ein paar Sekunden gebrochen werden. In ein paar Sekunden könnten Universitäten und Forschungsinstitute in die Privatsphäre von Menschen eindringen. In ein paar Sekunden könnte über den Sieg von Kriegen entschieden werden. Doch wir haben gesehen. In solch einer Situation haben die Kryptographen immer zurück geschlagen. Obwohl es noch keine Quantenrechner gibt: Die Quantenkryptographie ist der Quantenanalyse schon wieder einen Schritt voraus.

5.4 Quantenkryptographie

Stephen Wiesner hatte als Doktorand die Vision Quantengeld zu entwickeln; absolut fälschungssicheres Geld auf der Basis der Quantentheorie zu entwickeln.

Photonen im Raum schwingen in verschiedene Richtungen. Beispielweise können wir die vier Richtungen auf, ab, links und rechts unterscheiden. Wenn jetzt Photonen durch einen Polarisationsfilter gesandt werden, bleiben nur jene Photonen übrig, die in Richtung des Filters schwingen. Bei unserem Beispiel werden also bei einem vertikalen Filter die Photonen in Auf- und Abwärtsbewegung durchkommen.

Den Polarisationsfilter wollte Wiesner nun für Banknoten nutzen. Er schrieb hierfür eine offizielle Seriennummer auf die Banknote und in die Banknote kommen 20 Polarisationsfilter. Hinter den Filtern sind Photonen gespeichert. Ein Fälscher könnte jetzt beliebige Zuordnungen zwischen Seriennummer und Polarisationsfiltern machen, doch die Bank ist im Besitz einer Urliste. Stimmt die Zuordnung nicht überein, ist der Schein gefälscht. Dem Fälscher bleibt also nichts anderes übrig als Banknoten zu kopieren. Doch zum Kopieren muss der Fälscher die Polarisationsfilter und ihre Werte auslesen. Das stellt quantentheorisch eine unmögliche Aufgabe dar, wodurch das Kryptosystem sicher wird.

Wir definieren einen Filter mit der Konfiguration vertikal. Wir haben erwähnt, dass Photonen mit Auf- und Abwärtsbewegung durchkommen werden; links- oder rechtsschwingende nicht. Wir nehmen jetzt von der Anschauung Abstand, dass es sich um 4 verschiedene Zustände handle, sondern machen wir eine Einteilung in 360°. Wir nehmen wieder eine vertikale Konfiguration her und wann kommen Photonen jetzt durch den Filter? Bei 0° und 180° wird es ganz sicher gelingen. Wie schaut es mit 1° aus? Eventuell sind die Grenzen des Filters nicht genau geschliffen und lässt das Photon noch durch. Eventuell sind sie genau geschliffen und das Photon kann nicht durch kommen. Als Kon-

sequenz: Was sagt es uns, wenn es nicht den Spalt kommt? Wir wissen dann, dass das Photon nicht in der Konfiguration 0° oder 180° steht. Vom Rest wissen wir nichts. Und jetzt nehmen wir auch von Anschauung Distanz, dass es sich um 360 Schritte handle. Wenn wir den Filter auf Hunderttausend Schritte genau konfigurieren, dann haben wir quasi Hunderttausend unmessbare Probleme.

Wenn jetzt ein Fälscher die Banknote mit zahlreichen Photonen zupflastert, dann bleiben diese Photonen in der Banknote gefangen und die Bank erkennt, dass sie manipuliert wurde. Charles Bennett schaffte es diese Ideen auf die Quantenkryptographie zu übertragen. Wir nehmen an wir hätten Filter bzw. Polarisationen, die 8 Schritte beschreiben. Wir definieren also diagonale Filter in 2 Richtungen (also insgesamt 4) und vertikal-horizontale Filter in 2 Richtungen (insgesamt 4; $4+4=8$). Alice schickt zufällig irgendeine Reihenfolge von Photonen mit Polarisation. Sie selbst weiß ja, wie die Polarisation konfiguriert ist. Am anderen Ende sitzt Bob, der diese Folge empfängt. Er hat einen Filter, der nicht (wie bei Alice) 2 Zustände gleich behandelt, sondern 4 Zustände (also wenn bei Alice eine Diagonale eingezeichnet ist, dann hat Bob beispielweise ein Kreuz). Bob empfängt diese Folge und konfiguriert seinen Filter nach Zufall. Am Ende sagt Bob Alice über eine unsichere Leitung, was er gelesen hat. Dabei verrät er aber nicht die genaue Konfiguration (4 Zustände), sondern nur die 2 binären Werte. Alice sagt ihm daraufhin, welche Wert bei ihr ident sind und der Schlüssel ist vereinbart.

Der Trick bei der ganzen Sache ist die Zufälligkeit und die Unmöglichkeit der genauen Messung. Alice wählte zufällig Werte und Bob wählte (aus Sicht des Angreifers) zufällige Werte. Selbst wenn Eve die Daten lesen könnte... die Quanten verändern ihren Zustand durch das bloße Betrachten. Dadurch wird der Angriff auffällig. Die Idee ist genial. Die Idee wurde hier etwas knapp behandelt, aber Charles Bennetts Konzept müsste man etwas tiefer aufgreifen, damit man es mit Schulphysik erklären könnte. Das würde hier jedoch den Rahmen sprengen.

Auf jeden Fall mit der Quantentheorie, der Zufälligkeit und der Unmöglichkeit der genauen Messung können sich Bob und Alice einen Schlüssel ausmachen ohne dass jemand ihn mitbekommt. Niemandem ist es möglich den Schlüssel zu erfahren. Und mit diesem Schlüssel kann beispielweise das One-Time-Pad angewandt werden. Die perfekte Verschlüsselung, die niemals geknackt wird. Niemals? Niemals gibt es in der Kryptologie nicht. Aber das Problem bei der Quantenkryptographie ist es sowieso, dass man noch keine effizienten Quantencomputer bauen kann. Erst wenn das geschieht, wird das Interesse steigen und dann werden die Kryptoanalytiker und Kryptographen verstärkt auf diese Methode schauen und eventuell auch irgendwo einen technologischen Fortschritt ausnutzen, um das Verfahren zu brechen.

5.5 Rechtliche Aspekte

Die Welt ist einfach vor sich selbst nicht sicher. Menschen schaffen immer wieder geniale Konzepte, wie sie Codes knacken können. Dabei werden oft Methoden eingesetzt, die gar

nicht so genial sind, wie Nerds gerne hätten. Es werden sogenannte Non-IT Attacken entwickelt. Es geht also überhaupt nicht darum den Schlüssel zu knacken, sondern ein Lieferwagen wird vor Alices Haus geparkt und bei jedem Tastenanschlag sendet Alice elektromagnetische Wellen aus. Diese Wellen können hochsensibel gemessen werden und damit weiß der Computer, was Alice getippt hat. Bei dieser Art handelt es sich um nichts weiter als einen elektronischen Lauschangriff.

Eine andere Variante wäre Unterwanderung. Der Computer enthält bereits Software, die Tastenanschläge und gepflegte Kontakte aufzeichnet. Wenn Alice ihre geheime E-mail verschlüsselt, kennt das Programm die E-mail schon längst den Klartext. Alice schickt ihre E-mail hinaus und zugleich sendet das Programm die E-mail im Klartext an seinen Master. Die Verschlüsselung ist hier komplett sinnlos.

Inwiefern spielt das Recht jetzt eine Rolle? War es eine gute Idee die Schlüsselgröße für USA-Exporte zu regulieren? War es eine gute Idee RSAs Mächtigkeit zu limitieren?

Gegen Lauschangriff bietet der Handel Folien an, die in Wänden eingebettet elektromagnetische Strahlung um einen riesigen Faktor abschwächt und die Überwachung wird wesentlich schwieriger.

Linux-Entwickler haben sie zusammengeschlossen und eine Linux-Distribution geschaffen, die keine Angriffsfläche bietet. Der Staat kann keine Software im Hintergrund laufen lassen. Das System ist abgeschottet und in einer Umgebung, die es nicht mehr verlassen kann.

Man könnte an dieser Stelle argumentieren, dass 90% der Nutzer trotzdem weiterhin Windows gebrauchen werden und damit der Großteil der Nutzer (der Terroristen) abgedeckt werden kann. Falsch... weil in dem Moment, wo Terroristen eine Alternative sehen, die für sie einen Vorteil herbeiführt, werden sie die Alternative benutzen. Nur weil jemand

Kapitel 6

Anhang

.1 Fragen

- Gebe einen Überblick über die Einteilung der Kryptologie (Geschichte)
- Erkläre die Begriffe Fano-Bedingung und Präfixcode (Kodierungstheorie)
- Welche Grundproblematik behandelte die Kryptologie bis zum 20. Jahrhundert?
- Stärken und Schwächen von monoalphabetischer Substitution?
- Vergleiche Transposition und Substitution
- Stärken und Schwächen der polyalphabetischen Verschlüsselung
- Welche Anwendung findet XOR in der Mathematik/Informatik?
  Welche Verbindung besteht zwischen XOR und Modulo?
- Das One-Time-Pad ist bewiesen unknackbar. Welche Problematik hat es trotzdem?
- Methoden der klassischen Kryptoanalyse
- Was versteht man unter Stacks? Wie funktionieren Stacks?
- Wie werden Zufallszahlen erzeugt? Sind sie zufällig?
- Vor- und Nachteile der symmetrischer und asymmetrischer Verschlüsselung?
- Ähnlichkeiten zwischen Diffie-Hellman und Rivest-Shamir-Adleman
- Auf welcher Problematik basiert RSA?
- Welche Besonderheiten besitzen Primzahlen? Wieso eignen sie sich so gut?

- Wie wird Kryptographie heute eingesetzt? Betrifft es uns alle?
- Quantenrechner und Quantenkryptographie

10 Maturafragen. Versuche über jede Frage 10 Minuten lang zu referieren.

1. Alle erwähnten Verfahren bis 1950 erklären (Funktionsweisen)
2. Überblick über die Entwicklung, Methoden der Kryptographie (mono/polyalphabet, digital, beteiligte Wissenschaften ...)
3. Überblick über Methoden der Kryptoanalyse (Kasiski, Häufigkeit, Friedman)
4. Diffie-Hellman-Schlüsselaustausch (Funktionsweise, Beweis, Angriff)
5. symmetrisch vs. asymmetrisch (analoge, reale Beispiele, Vor-/Nachteile)
6. Pseudozufallszahlen (Schieberegister)
7. Theorie hinter Modulo (Rest, Restklassen, zB Uhren, Informatik)
8. Theorie hinter Primzahlen (unvorhersehbar, unendlich, Eratosthenes, Probedivision)
9. Nach- und Vorgeschichte von RSA (GCHQ, PGP/GPG, Phil Zimmermann, USA)
10. Technische & Gesellschaftliche Auswirkung von Kryptographie (Web of Trust, Authentifikationen, neue Einsatzgebiete, SSL, Integrität, Hashes/Fingerprints)

.2 Glossar

Diese Liste kann zum Eigentest verwendet werden. Lest das erste Wort und gebt selbst eine Definition an. Überprüft danach mit dieser Liste.

Nicht alle Kryptosysteme sind hier notiert.

Algorithmus Schrittweise Anleitung zur Lösung eines Problems

Kryptosystem Ein kryptologisches Verfahren, welches das Verschlüsseln und Entschlüsseln von Inhalten erlaubt

Kodierung Eindeutige Zuordnung der Elemente der Menge A zu den Elementen der Menge B

Involution Algorithmen zu Ver- und Entschlüsseln sind ident (siehe XOR, ROT)

Steganographie Versucht Inhalte durch Verschleierung geheim zu halten

Kryptographie Versucht Inhalte geheim zu halten, indem sie unverständlich gemacht werden

Monoalphabet(ik) Das Kryptosystem benötigt bloß ein einziges Alphabet

Polyalphabet(ik) Das Kryptosystem benötigt mehrere Alphabete

Substitution Ein Element wird durch ein anderes ersetzt

Transposition Ein Element tauscht mit einem anderen Element die Position

Präfixcode Kodierung, die die Fano-Bedingung erfüllt

Polybios-Matrix Buchstaben werden in einem Rechteck zusammengefasst und dann (beispielweise) spaltenweise statt zeilenweise gelesen

Emphemeralschlüssel Einmalig verwendeter Schlüssel

security by obscurity Sicherheit wird durch Geheimhaltung des Algorithmus' angestrebt

Morsecode Alphabet mit Licht- oder Stromsignalen zur Kommunikation (heute vorwiegend Seefahrt)

Binär Alphabet mit nur 2 Zuständen. Mit 0/1 oder wahr/falsch notiert

Hexadezimal Alphabet mit 16 Buchstaben. In der üblichen Schreibweise werden die Zahlen 0–9 durch A–F erweitert.

boolsche Algebra Befasst sich mit Operationen in der Logik (UND, ODER, XOR)

beatnik Eine esoterische Programmiersprache

Skytale Umfang einer Skytala

One-Time-Pad Das erste bekannte – mathematisch bewiesen – sichere Kryptosystem

Häufigkeitsanalyse Textanalyse mittels der Anzahl der Vorkommen eines Buchstabens

Kasiski-Test Analysiert polyalphabetischen Code nach Wiederholungen

Friedman-Test bietet Formeln zur Messung der Häufigkeitsverteilung von Buchstaben in einem verschlüsselten Text

Kerckhoffs Kryptologe, der Prinzipien für Kryptosysteme definierte. Am wichtigsten: Sicherheit darf nicht von der Geheimhaltung des Algorithmus' abhängen

Schlüsselproblem Es gibt sichere symmetrische Verfahren, jedoch muss der Schlüssel zur gemeinsamen Kommunikation ausgetauscht werden

asymmetrische Verschlüsselung Ver- und Entschlüsselung erfolgt mit unterschiedlichen Schlüsseln

Public/Private-Key Schlüssel zur asymmetrischen Verschlüsselung

Bob, Alice and Eve Alice sendet Bob eine Nachricht und Eve greift an (zur Simulation von Kryptosystemen verwendete Namen)

GCHQ Abteilung des Britischen Geheimdiensts, die die RSA-Verschlüsselung ein paar Jahre vor Rivest, Shamir und Adleman begründeten

Pseudozufallszahlen Annähernd zufällige Zahlen

Binomialkoeffizient mathematisches Element aus der Kombinatorik

Primzahlen Zahlen, die nur durch 1 und sich selbst teilbar sind

Modulo Rest einer Division

Restklasse Folge von Zahlen, die alle den selben Rest beßuglich einer Division aufweisen

Modulare Arithmetik befasst sich mit dem Verhalten von Zahlen in Restklassen

Square and Multiply schnelles Verfahren zur Lösung von Gleichungen der Form $a^c \bmod m$

Eulersche Funktion Gibt die Anzahl der teilerfremden Zahlen zu n zurück

Sieb des Eratosthenes Erzeugt schnell eine große Liste mit Primzahlen

Probedivision ein bekanntes Faktorisierungsverfahren

Modulares Inverses Zahl, die eine Gleichung der Form $a \cdot c \bmod p \equiv 1$ erfüllt (a und p vorgegeben, c ist das modulare Inverse von a)

RSA Kryptosystem, welches aktuell am meisten für asymmetrische Verschlüsselung eingesetzt wird

PGP/GPG Implementierungen von RSA für Normalanwender

Authentizität Sicherheit, dass das Gegenüber jener ist, der er vorgibt zu sein

Hashes Einwegfunktionen. Wird gebraucht, um Texte auf Ähnlichkeit zu prüfen

Fingerprint Hash für RSA-Schlüssel

Quantenkryptographie Versucht Kryptographie auf Basis der Quantenphysik zu ermöglichen

.3 Wichtige Kryptologen

Hier sind nur jene zu finden, die besprochen wurden oder heute aktuell wichtig sind.

Charles Babbage (* 1791 † 1871) siehe Vigenère

Ada Lovelace (* 1815 † 1852) siehe Vigenère

Georges Jean Painvin (* 1886 † 1980) siehe ADFGVX

Friedrich Wilhelm Kasiski (* 1805 † 1881) siehe Kasiski-Test

William Friedman (* 1891 † 1969) siehe Friedman-Test

Auguste Kerckhoffs (* 1835 † 1903) siehe Kerckhoffs' Gesetze

Irving John Good (* 1916 † 2009) Turing-Bombe, siehe Enigma

Alan Turing (* 1912 † 1954) siehe Bletchley Park, Turing-Bombe, Komplexitätstheorie

Martin E. Hellman (* 1945) siehe Diffie-Hellman-Verfahren

Whitfield Diffie (* 1944) siehe Diffie-Hellman-Verfahren

James H. Ellis (* 1924 † 1997) siehe GHCQ-Geschichte

Clifford Cocks (* 1950) siehe GHCQ-Geschichte

Malcolm J. Williamson (unbekannt) siehe GHCQ-Geschichte

Leonard Adleman (* 1945) siehe RSA

David Kahn (* 1930) schrieb das Standardwerk "The Codebreakers" über die Geschichte der Kryptologie

Bruce Schneier (* 1963) Autor von "Applied Cryptography", international bekannter Computersicherheitsexperte

.4 Status des Dokuments

PDF Kryptologie 3.0-090617 "public complete"

Erweiterbare Kapitel:

- graphische Beschreibung der Kryptosysteme
- Mathematische Betrachtung eines Fixpunktes
- Alan Turing und ENIGMA (geordneter)
- Hashes allgemein (ausbauen)
- Authentifizierung & Co. (zumindest bei RSA kurz anschneiden)
- Determinierung, deterministische Pseudozufallsgeneratoren, CSPRNG, Mersenne-Twister, Abzählreim
- Glossar und Fragen

Fehlende Kapitel:

- Playfair (Polygrammsubstitution)
- homophone Reihen
- Friedman-Test ausbauen
- Autokorrelation (XOR-Wette)
- Sieb des Atkin
- Pseudoprimzahlen
- Satz von Euler
- AKS-Primzahltest
- Kleiner Satz von Fermat, Pseudoprimzahlen, Carmichaelzahl
- Mustersuche
- Heron Verfahren
- (Open)PGP
- Fermatscher Primzahltest
- Rubberhose cryptanalysis
- Feistelchiffre
- SUBSET-SUM

- Rabin-Kryptosystem
- Miller-Rabin Test
- Cryptool, crypto, python-crypto
- UNIX und salts, SSL, base64
- Blowfish, IDEA, SHA, MD5, DES, Twofish, RC4, AES, Cast128
- Alan Mathison Turing – Leben (+ Turingmaschine)

Fehlende Implementierungen:

- Freimaurerchiffre (Darstellungsproblem)
- ENIGMA (Modellproblem, Komplexität/Aufwand)
- Steganographie (PHP-GD) (unvollständig)
- Wörterbuch legt Relationen zwischen Buchstaben fest (abc = a[+1][+2] = 12) im Wörterbuch nach Äquivalenten suchen

.5 Autor

http://lukas-prokop.at/

Mein Name ist Lukas Prokop und ich entschied mich das Thema *Kryptologie – Eine verschlüsselte Wissenschaft* als fächerübergreifendes Spezialgebiet für meine AHS-Matura am BRG Viktring-Klagenfurt zu wählen. Von Anfang an war mir klar, dass ein sehr ausführliches Spezialgebiet zu meinen Maturaaufgaben zählt. Schließlich lernte ich extra dafür ab der 7. Klasse LaTeX 2_ε kennen, um dabei etwas professioneller zu wirken ;-)

Ich entschied mich mein fächerübergreifendes Spezialgebiet in Fächern zu schreiben, die zu meinen Lieblingen zählen. Mathematisch betrachtet ist RSA sicher ein gutes Thema und gewisse Themen wie Modulorechnungen kannte ich bereits von jahrelangen Programmiererfahrungen. Deshalb war der mathematische Teil auf RSA konzentriert und für Informatik wollte ich eine Implementierung bauen (in python, welches ich auch ab der 7. Klasse lernte). Eine saubere Implementierung würde wohl Jahre dauern, aber zumindest die Grundkonzepte sollten in der Programmierung ersichtlich sein. Mein Bruder leihte mir zudem Simon Singhs *Geheime Botschaften*. Dort wurde die Kryptologie zwar weder mathematisch noch informationstechnisch betrachtet, aber der Geschichte der Kryptologie widmete ich deshalb den ersten Teil des Spezialthemas. Diesem Dokument liegen einige cryptotools bei, in denen die Kryptosysteme implementiert wurden, die hier auf zahlreichen Seiten vorgestellt wurden. Auch RSA.

.6 Attachments

http://lukas-prokop.at/proj/spezialgebiete/kryptologie.pdf
http://lukas-prokop.at/proj/spezialgebiete/kryptologie.tar.gz
http://lukas-prokop.at/proj/spezialgebiete/tools.tar.gz

.7 Dank

Prof. Schmidhofer (Informatik) und Prof. Egger (Mathematik) vom BRG Viktring
Auch Dank an meine Brüder Christoph (Buch leihen) und Michael (Erfahrungsbericht über Keysigning Parties)
Und liebe Grüße an meinen Parallelklassler
written with LaTeX 2_ε

.8 Copyright

Ich habe keinerlei Problem damit, wenn Inhalte ohne Einschränkungen kopiert, verteilt und verändert werden (auch ohne Namensnennung). Ich selbst habe die Inhalte auch nur durch zahlreiche Quellen erhalten. Angefangen in der Schule wo man rechnen, schreiben und lesen lernt. Wissen ist dazu da, geteilt und entwickelt zu werden.

© Lukas Prokop 2009

Literaturverzeichnis

[1] Beutelspacher, Albrecht: *Kryptologie*. vieweg, 1987.

[2] Chang, Gosling: *US Patent 5724425 Appendix*. freepatentsonline, 1998.

[3] Diffie, Whitfield und Martin E. Hellman: *New Directions in Cryptography*. Invited Paper, http://www.cs.jhu.edu./ rubin/courses/sp03/papers/diffie.hellman.pdf, 1976.

[4] Helmut Herold, Bruno Lurz, Juergen Wohlrab: *Grundlagen der Informatik*. Pearson Studium, 2007.

[5] Müller, Winfried: *Kryptographie – Wie schützt man elektronische Daten?* Vortrag beim Treffpunkt Mathematik http://tom.pi-ahs.at/, 2009.

[6] Singh, Simon: *Geheime Botschaften*. dtv, 2001.

[7] Taschner, Rudolph: *Mathcast*.
mathcast Homepage: http://mathcast.org/, 2008.

[8] Wikipedia: *Häufigkeitsverteilung der deutschen Buchstaben*.
http://de.wikipedia.org/wiki/Deutsches_Alphabet.

[9] Wikipedia: *Wikipedia Kategorie Kryptologie*.
http://de.wikipedia.org/wiki/Kategorie:Kryptologie.

[10] Zimmermann, Philip R.: *Why I wrote PGP*. PGP User's Guide, 1991 updated 1999.