



universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit
Kryptographie - Prinzipien, Einführung, Anwendung und
gesellschaftliche Relevanz mit Schulbezug

Verfasser:
Stefan Gruber

angestrebter akademischer Grad
Magister der Naturwissenschaften (Mag. rer.nat)

Wien, 2010

Matrikelnummer: 0501370

Studienkennzahl lt. Studienblatt: A190 412 884

Studienrichtung lt. Studienblatt: UF Physik und UF Informatik und Informationsmanagement

Betreuer: Mag. Dr. Andreas Ulovec

Zusammenfassung

Wenn zwei Menschen miteinander kommunizieren, dann möchten sie in der Regel nicht, dass dritte wissen um was es in der Unterhaltung geht. Seitdem Unterhaltungen nicht mehr von Angesicht zu Angesicht, sondern auch über große Distanzen stattfinden können, versuchen wir den Inhalt unserer Kommunikation möglichst gut zu verschleiern. Die Wissenschaft, die sich mit Methoden dieser Verschleierung befasst, wird *Kryptographie* genannt. Schon zu Zeiten der Antike wurde in diesem Gebiet geforscht. Spätestens seit Einführung der Funktechnologie sind gute kryptographische Systeme zentrale Dreh- und Angelpunkte in bewaffneten Konflikten. Doch auch im privaten und zivilen Bereich ist durch den Aufstieg des Internets Geheimhaltung von Kommunikation nicht mehr wegzudenken. Aus diesem Grund wollte ich in dieser Arbeit einen guten Überblick und ersten Einstieg in die Kryptographie geben, immer mit dem Gedanken, diesen oft sehr mathematischen Stoff in der Schule vermitteln zu wollen. Der Hauptteil dieser Arbeit beschäftigt sich mit diversen Algorithmen, die zum gegenwärtigen Zeitpunkt Anwendung finden. Eine Einteilung in symmetrische und asymmetrische Verfahren teilt die Fülle an Algorithmen zunächst in zwei Lager, die jeweils sehr verschiedene Ansätze haben, was ihre Funktion betrifft. Bei den asymmetrischen ist es so, dass mathematische Probleme zum Ver- und Entschlüsseln ausgenutzt werden. Bei symmetrischen hingegen werden mehrere, raffinierte Schritte wie z.B. Verschiebung, Austauschen, Permutation usw. dazu verwendet, bekannten Angriffen entgegenzuwirken. Durch die Analyse der einzelnen Algorithmen und die einfachere Formulierung soll ein Grundverständnis für ihre Arbeitsweise vermittelt werden. Dabei ist die Formulierung an die entsprechenden Werke und Dokumente angelehnt, die zu dem jeweiligen Verfahren verfügbar sind (Quellenangaben sind im jeweiligen Abschnitt bzw. Absatz zu finden). Teils sind dies Sammelwerke zu verschiedenen Themen der Kryptographie, Teils Werke zu speziellen Themen und Teils Dokumente der US Regierung, die ohne Anspruch auf Urheberrecht, das heißt gemeinfrei, der Öffentlichkeit zugänglich gemacht wurden. Zu den klassischen Arten von Kryptographieverfahren, gesellt sich eine neue, die Quantenkryptographie. Im entsprechenden Kapitel wird die physikalische Grundlage hierfür erklärt und das bekannteste Protokoll zur Übermittlung von Nachrichten erläutert. Außerdem werden Beispielanwendungen angeführt, um den praktischen Umgang mit kryptographischer Software zu erlernen. Zusätzlich findet man in der Arbeit einen Überblick über die geschichtliche Entwicklung der Kryptographie mit besonderem Augenmerk auf Alan Turing, der so viel für die Mathematik, Informatik und viele andere Disziplinen der

Wissenschaften beigetragen hat. Das letzte Kapitel der Arbeit befasst sich speziell mit dem Unterricht und beinhaltet mögliche Inhalte für den Informatikunterricht.

Inhaltsverzeichnis

1	Allgemein	1
1.1	Was ist Kryptographie?	1
1.2	Wozu braucht man Kryptographie?	1
1.3	Ein kurzer geschichtlicher Werdegang	2
1.3.1	Die Enigma und Alan Turing	4
1.4	Ziele der Kryptographie	7
1.5	Arten von Kryptosystemen	8
1.6	Kryptographie in der Schule	9
1.7	Vertrauen in kryptographische Systeme	10
2	Asymmetrische Kryptographieverfahren	15
2.1	Use Cases	16
2.2	RSA	16
2.2.1	Schlüsselerzeugung, Ver- und Entschlüsselung	16
2.2.2	Software	18
2.2.3	Unterrichtssequenz	21
2.3	DSA	29
2.3.1	Schlüsselerzeugung, Signieren und Überprüfen	30
2.4	Elliptic Curve Cryptography	31
2.4.1	Schlüsselerzeugung	33
2.4.2	Ver- und Entschlüsselung	33
2.4.3	Unterrichtssequenz	33
3	Symmetrische Kryptographieverfahren	37
3.1	DES	38
3.1.1	Funktionsweise von DES	40
3.2	AES	45
3.2.1	Funktionsweise von AES	46
3.2.2	Unterrichtssequenz	51
3.3	Twofish	55
3.3.1	Funktionsweise von Twofish	55
3.3.2	Unterrichtssequenz	59
3.4	One-Time-Pad	62
3.4.1	Funktionsweise und Anwendung	63

3.4.2	Unterrichtssequenz	66
3.5	Solitaire	66
3.5.1	Funktionsweise von Solitaire	67
3.5.2	Unterrichtssequenz	72
4	Quantenkryptographie	73
4.1	Quantenmechanik	74
4.1.1	Wellenphänomene	74
4.1.2	Der fotoelektrische Effekt, Teilchenphänomene	76
4.1.3	Teilchen als Wellen	78
4.1.4	Schrödinger-Gleichung	78
4.2	Heisenberg'sche Unschärferelation	80
4.3	Photonen	80
4.4	BB84-Protokoll	81
5	Unterricht	83
5.1	Banken	83
5.2	WLAN	84
5.3	Passwörter	85
	Literatur	95
	Lebenslauf	97

1 Allgemein

1.1 Was ist Kryptographie?

Kryptographie ist die Wissenschaft, die sich mit der Verschlüsselung von Informationen befasst. Man versucht eine Folge von Zeichen nach einem bestimmten Verfahren so zu verändern, dass das Resultat nicht mehr interpretierbar ist. Um die ursprüngliche Zeichenfolge wieder zu erhalten, muss man das passende Gegenverfahren anwenden. Bei diesen Verfahren werden Geheimnisse so verwoben, dass idealer Weise nur die Kenner des Geheimnisses und des Verfahrens aus der veränderten wieder die originale Zeichenfolge extrahieren können. Die originale Zeichenfolge nennt man *Klartext*, die veränderte *Geheim-* oder *Ciphertext*. Das angewendete Geheimnis ist der *Schlüssel* und das Verfahren nennt man *Verschlüsselung*. Das Gegenverfahren zur Verschlüsselung ist die *Entschlüsselung*. Die Zeichenfolge ist beliebig. Es kann Text, Bild, Ton oder sonst etwas sein. Wenn man Kryptographie mit Computern betreibt, handelt es sich dabei ohnehin lediglich um Bitfolgen. [12]

1.2 Wozu braucht man Kryptographie?

Nicht nur in unserer modernen Zeit, sondern auch schon im Altertum, wollte man über große Distanzen Informationen sicher übermitteln. Ein römischer General wollte zum Beispiel seinen Offizieren die nächsten strategischen Züge mitteilen, ohne selbst zu jedem hin reiten zu müssen. Er musste somit Boten schicken. Das Problem mit den Boten ist, dass man ihnen zum einen nicht voll und ganz vertrauen kann, es könnten ja auch feindliche Spione sein, zum anderen können sie gefangen genommen und verhört werden. Der Bote sollte also nicht wissen, was in der Nachricht, die er übermitteln soll steht. Die Offiziere sollen zudem sicher sein können, dass der Befehl tatsächlich vom General kommt und nicht von den Feinden, die sie eventuell in einen Hinterhalt locken wollen.

Große Distanz kann aber auch große *zeitliche* Distanz sein. Zum Beispiel haben Geheimdienste großes Interesse daran, dass Informationen zu diversen Operationen möglichst lange möglichst sicher sind. Man möchte also erreichen, dass selbst wenn jemand die verschlüsselten Informationen bekommt, dieser jemand sie über sehr lange Zeit, am besten nie, aber zumindest mehrere Jahrzehnte, keine Möglichkeit hat den Klartext zurück zu rechnen.

Heute wird Kryptographie natürlich immer noch militärisch eingesetzt, im zivilen

Bereich ist sie aber eben so wenig weg zu denken. Wenn ich zum Beispiel mit jemandem über mein Mobiltelefon telefoniere, will ich nicht, dass jemand im Nebenraum die Übertragung mitschneidet. Gegen einen direkten Lauschangriff werde ich nur schwer etwas tun können, aber ich *kann* etwas tun. Leise sprechen oder in einen abhörsicheren Raum gehen. Wenn jedoch die Funksignale ohnehin nicht verschlüsselt sind, bringt mir das auch nichts.

1.3 Ein kurzer geschichtlicher Werdegang

Wie bereits im vorherigen Abschnitt erwähnt, hatten schon Menschen im Altertum das Verlangen bestimmte Daten zu verschlüsseln. Eines der ältesten Verschlüsselungsverfahren waren die *Skytale*. Diese wurden bereits vor mehr als 2500 Jahren von den Spartanern verwendet. Hierfür wird ein Pergamentband oder ein Lederstreifen spiralförmig über die Skytale, ein zylindrischer Holzstab, gewickelt. Die Botschaft wird längs des Stabs auf das Band geschrieben. Wenn man das Band nun abwickelt, stehen die Buchstaben scheinbar in willkürlicher Anordnung auf dem Band. Um die Botschaft zu lesen, benötigt man eine Skytale mit demselben Durchmesser wie die des Absenders. Der Durchmesser der Skytale ist somit der geheime Schlüssel.

Ein weiteres Verfahren, nämlich das vom Griechen Polybius entwickelte „Polybius Quadrat“ zeigt zwei wesentliche Schritte in der Entwicklung von Ciphern. Hier werden die Buchstaben des Alphabets in ein 5×5 Quadrat geschrieben und dann jeder Buchstabe durch seine Koordinaten identifiziert. (Siehe Tabelle 1). Da unser modernes Alphabet 26 Buchstaben hat, müssen sich i und j ein Kästchen teilen. Durch diese Überlegung, konnten Nachrichten dadurch übermittelt werden, indem man in jeder Hand die richtige Anzahl an Fackeln nahm und diese hoch hielt. Will man zum Beispiel den Buchstaben „h“ übermitteln, hält man in der rechten Hand drei und in der linken zwei Fackeln.

Tabelle 1: Polybius Quadrat. [5]

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	ij	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

Eine der bekanntesten Arten aus antiken Zeiten ist die sogenannte Caesar-Verschiebung.

Benannt nach seinem berühmtesten Anwender, Julius Caesar, stellt dieses Verfahren einen Vertreter der Verschiebeschiffren dar. Julius Caesar schrieb das Alphabet zweimal untereinander, beim zweiten Mal begann er jedoch mit D. Anschließend wurde jeder Buchstabe aus dem obigen Alphabet durch den des unteren ersetzt. A mit D, B mit E usw. [5] Natürlich kann man die Anzahl der Plätze, um die verschoben wird variieren, oder das untere verkehrt herum aufschreiben, oder überhaupt in keiner vernünftigen Reihenfolge. Man kann das Plain Text-Alphabet somit durch $26! - 1 = 4 \cdot 10^{26}$ andere Alphabete austauschen, was gar kein so kleiner Schlüsselraum ist. Leider hat diese Art von Chiffre aber eine Schwachstelle, die sie extrem leicht knackbar macht. Wenn man die Sprache, in der der Text verfasst ist, erraten kann, muss man nur die Häufigkeit, mit der die Buchstaben vorkommen erfassen und dann mit der Häufigkeit in der jeweiligen Sprache vergleichen. Wenn im Ciphertext der Buchstabe R am häufigsten vorkommt, wird er sehr wahrscheinlich dem häufigsten Buchstaben in der vermuteten Sprache entsprechen, bei Deutsch wäre das E. Selbst wenn man nur Papier und Bleistift als Hilfsmittel hat, ist das in recht überschaubarer Zeit machbar. Mit Computern und Programmen wie OpenOffice.org Calc oder MS Office Excel dauert das dann nicht viel länger, als den Ciphertext abzutippen.

Mit dem Ersten Weltkrieg begann sich eine immer größere Notwendigkeit für sichere kryptographische Verfahren abzuzeichnen. Man wollte einerseits nicht auf die Vorteile von neuer Funktechnologie verzichten, andererseits war man sich mehr als bewusst, dass diese Form der Übertragung wesentlich leichter abgehört werden kann, als Überlandleitungen oder gar Botendienste. Die Deutschen hatten ein Verfahren namens ADFGX, von dem sie überzeugt waren, es sei unknackbar. Gerade rechtzeitig für eine Offensive gegen Frankreich, wurde ADFGX als sicherer Cipher angenommen und als unersetzlich für das Gelingen der folgenden militärischen Operationen angesehen, da eine Geheimhaltung der genauen Positionen der deutschen Truppen und der Befehle das Überraschungsmoment sichern sollten. Im Juni 1918 waren die Deutschen nur noch 100 km von Paris entfernt. Die Franzosen mussten sich auf ihre Geheimwaffe verlassen. Diese war ein französischer Geologe namens Georges Painvin, der ein Talent für Kryptoanalyse hatte. Beim Versuch ADFGX zu knacken, verlor er durch tage- und nächtelange Arbeit 15 kg, jedoch nicht vergebens. Er konnte ADFGX knacken und wichtige Botschaften entschlüsseln aus denen man die Position der Deutschen erfuhr. Alliierte Truppen wurden entsandt und die Deutschen in einer fünftägigen Schlacht abgewehrt. [15]

Neben ADFGX wurden im Ersten Weltkrieg hauptsächlich Codebücher und andere,

ebenfalls manuell auszuführende Verfahren, verwendet. Codebücher sind Bücher, mit denen jedes Wort eines Textes in ein unverständliches anderes Wort oder eine Zahl umgewandelt werden konnte. Erst durch die schlechten Erfahrungen mit dieser Art von Verfahren kamen im Zweiten Weltkrieg vermehrt durch Maschinen automatisierte Verschlüsselungen zum Einsatz. Die bekannteste dürfte wohl die deutsche Enigma sein (siehe Abschnitt „Enigma und Alan Turing“). Bis 1970 dominierten diverse Chiffriermaschinen den Markt, die zum Teil für die damalige Zeit beachtliche Sicherheit bieten konnten. Seit ungefähr 1970 übernahmen diese Aufgaben die allgemeinsten aller Maschinen, Turingmaschinen in Form von elektronischen Computern. [19]

1.3.1 Die Enigma und Alan Turing

Alan Mathison Turing wurde am 23. Juni 1912 in London geboren. Nach seiner schulischen und hochschulischen Ausbildung in Mathematik auf dem Kings College in Cambridge, graduierte er 1934. Zwei Jahre später veröffentlichte er sein wichtigstes Werk, „On Computable Numbers, with an Application to the Entscheidungsproblem“. In diesem beschrieb er den abstrakten, digitalen Computer, den man heute Universale Turing-Maschine bezeichnet. Nachdem er seinen Ph.D. auf der Princeton Universität in den USA abgeschlossen hatte, wurde er ein Jahr nach seiner Rückkehr nach England in das Hauptquartier der Government Code and Cypher School nach Bletchley Park berufen, um von dort aus im Krieg gegen Nazi-Deutschland behilflich zu sein. Nach dem Krieg widmete er sich der Entwicklung von elektronischen, digitalen Computern. Im März 1951 wurde er zum Fellow of the Royal Society of London gewählt, eine der höchsten Ehren in Großbritannien und im März 1952 wegen homosexueller Aktivitäten angeklagt und zu zwölfmonatiger Hormon-„Therapie“ verurteilt. Von dem Land, zu dessen Rettung er maßgeblich beigetragen hat, so schandvoll behandelt zu werden, ertrug er mit amüsiertes Standhaftigkeit, indem er den durch die Hormone hervorgerufenen körperlichen Veränderungen mit Humor begegnete. Am 7. Juni 1954 beging er wahrscheinlich durch einen mit Cyanid vergifteten Apfel Selbstmord. Vermutlich wegen einer durch die Therapie verursachten Depression.

Schon einige Monate vor dem Ausbruch des Zweiten Weltkriegs, begann Alan Turing den Kampf mit der Enigma. Die Enigma war eine von den Deutschen eingesetzte Chiffriermaschine, die aber bereits kommerzielle Vorläufer seit 1923 hatte, die vom Berliner Arthur Scherbius entwickelt wurden. Die von den deutschen eingesetzten Versionen der Enigma waren selbstverständlich erweitert und schwerer zu knacken, als die kommerziell erhältlichen.

Am Tag nach der Kriegserklärung von Chamberlain an Nazi-Deutschland, am 4. September 1939, bezog Turing sein Quartier im Hauptquartier der *Government Code and Cypher School* (GC&CS), Bletchley Park. Dort wurden ab 1942 sogenannte „bombes“ gebaut. Maschinen, die dazu verwendet wurden, Enigmanachrichten zu entschlüsseln. Die bombes wurden von Turing entworfen und von Harold Keen gebaut. Diese Maschinen entschlüsselten pro Monat circa 39000 Enigmanachrichten. Zurückblickend betrachtet haben diese Anstrengungen den Krieg um gute zwei Jahre verkürzt.

Von 1926 an haben nach und nach mehr deutsche Militäreinheiten die Enigma übernommen. 1930 wurde das *Steckerbrett* hinzugefügt und 1935 wurde die Enigma schließlich auch von der Luftwaffe übernommen. Immer wenn man von „*der Enigma*“ spricht, meint man eigentlich eine Art von Chiffriermaschine und nicht eine spezielle Ausprägung davon. Denn es gab mehrere Variationen.

Der grundlegende Aufbau war folgender: Drei, später vier, Rotoren, eine Tastatur mit 26 Tasten, ein Brett mit 26 Lämpchen mit Buchstaben und ein Steckerbrett. In Abb. 1 sieht man einen schematischen Aufbau.

Wenn man einen Klartextbuchstaben drückt, leuchtet das entsprechende Ciphertext-Lämpchen auf. Der Ciphertext wurde dann per Funk, Morse oder Überlandleitungen übertragen. Jedes Mal, wenn eine Taste gedrückt wird, drehen sich im Inneren der Enigma ein oder mehr Zahnräder, was wiederum den Schaltkreis zu den Lämpchen veränderte und somit für jedes Drücken eines Buchstabens eine neue Konfiguration einstellte. Wenn man wiederholt ein und denselben Buchstaben drückte, wurde für jeden ein anderer Cipherbuchstabe erzeugt. Außer der Klartext-Buchstabe selbst. Ein Buchstabe wurde nie mit sich selbst verschlüsselt. Dies wurde natürlich von den Codebreakern in Bletchley Park ausgenutzt.

Im Inneren der Enigma sind zumindest drei Rotoren, auf denen jeweils 26 Buchstaben sind. Durch drei Fenster kann pro Rad ein Buchstabe eingestellt werden. Die Konfiguration der Buchstaben gab das „message setting“ wider.

Wenn sowohl der Sender, als auch der Empfänger das gleiche Setting eingestellt hatten, war die Verschlüsselungsoperation umkehrbar. Das heißt, dass beim Drücken der Ciphertext-Buchstaben die Klartext-Buchstaben aufleuchteten.

Zusätzlich musste das Steckerbrett auch die gleiche Konfiguration aufweisen.

Später wurden immer ausgeklügeltere Versionen der Enigma entwickelt. Ab 1941 gab es Enigmas mit einem vierten Rad. Außerdem konnte die Position der Räder vertauscht werden. Es gab auch nicht nur die drei bzw. vier eingebauten Räder, sondern

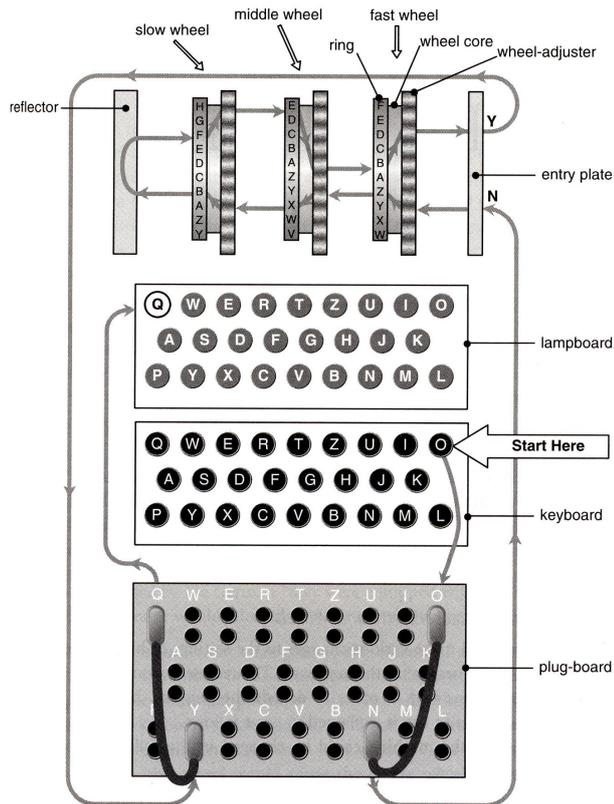


Abbildung 1: Schematischer Aufbau einer Enigma mit drei Rotoren und Steckerbrett.
(Mit freundlicher Genehmigung von Jack Copeland aus [1] entnommen.)

bis zu acht, die in beliebiger Reihenfolge eingebaut werden konnten.

Das rechteste Rad drehte sich bei jedem Tastendruck um einen *Klick* weiter. Nach einer bestimmten Anzahl an Klicks wurde das mittlere Rad um einen Klick weiter gedreht, welches dann wiederum nach einigen Klicks das rechteste weiter drehte. Wann welches Rad weiter gedreht wurde, hing davon ab, wie die Einkerbungen der Räder waren. Manche Einheiten hatten zwei Einkerbungen pro Rad, andere nur eine. Der Schlüssel für eine Übertragung war also weit mehr als nur das Setting der drei Rotoren zu Beginn. Er setzte sich zusammen aus der Einstellung der drei, bzw. vier Rotoren, der Konfiguration des Steckerbretts, der Auswahl der drei aus acht Rotoren, die Anordnung der drei Rotoren und welche Rotoren zur Auswahl standen. Dadurch waren Nachrichten auch dann „sicher“, wenn die Feinde eine Enigma ergatterten konnten. Welche der zahlreichen Kombinationen zu verwenden war, wurde mittels ausgedruckten Tabellen festgelegt. Sender und Empfänger hatten je ein Exemplar für einen Monat, in dem für jeden Tag die Konfiguration nachschlagen wurde.

Eine Konfiguration für einen Tag könnte zum Beispiel so aussehen: Die Radanordnung ist VI/II/IV (die Räder waren von I bis VIII durchnummeriert). Als nächstes braucht man das Setup für die Stecker. Zum Beispiel wäre B/R, U/Z, J/E, O/W, T/V und M/D möglich. Das heißt, dass man sechs Doppelstecker verwendete, wobei der erste B mit R, der zweite U mit Z usw. verband. Die Stecker haben gleichzeitig aber auch R mit B, Z mit U usw. verbunden, was sich später als ungünstige Vereinfachung herausstellen sollte.

Die dritte Komponente des Schlüssels war die Ringstellung. Jedes Rad hatte mittig einen Ring mit den 26 Buchstaben des Alphabets. Jeder Ring konnte auf einen Buchstaben eingestellt werden und positionierte somit das Rad auf eine bestimmte Weise. Eine Konfiguration könnte zum Beispiel RGY lauten. [1]

1.4 Ziele der Kryptographie

Das erste Ziel der Kryptographie ist es, *Vertraulichkeit* zu schaffen. Man möchte erreichen, dass nur dazu berechtigte Personen in der Lage sind, die verschlüsselten Informationen wieder zu entschlüsseln. Zum Beispiel möchte ich unter Umständen nicht, dass jemand meinen E-Mail-Verkehr mit meiner Tante mitlesen kann. Ich könnte mich also mit meiner Tante auf einen Kaffee treffen wo wir uns vertraulich auf ein gemeinsames Geheimnis und ein Verschlüsselungsverfahren einigen. Wenn sie mir dann eine E-Mail schreibt, deren Inhalt nach der Vereinbarung verschlüsselt ist, kann sie sehr sicher sein, dass nur ich die E-Mail lesen werde.

Ein weiteres Ziel ist die *Integrität* von Daten zu gewährleisten. Das heißt, dass man mit hoher Sicherheit sagen kann, ob die Daten nachträglich verändert wurden. Um beim vorherigen Beispiel zu bleiben: Wenn ich eine E-Mail von meiner Tante erhalte, möchte ich auch sicher sein, dass der Text den ich lesen werde, auch so von ihr verfasst wurde. Es soll nicht möglich sein, dass ein Dritter den Text abfängt und Teile herausnehmen oder hinzufügen kann, ohne, dass ich den Angriff mitbekommen werde. Aber nicht nur gegen gezielte Angriffe soll die Gewährleistung der Integrität der erhaltenen Informationen schützen, auch technische Fehler wie eine gestörte Übertragung sollen erkannt werden können.

Außerdem möchte man *Authentizität* der Daten erreichen. Ich will also überprüfen können, ob die E-Mail die ich von meiner Tante erhalten habe, auch wirklich von ihr verfasst und abgesendet wurde. Wenn sich jemand als meine Tante ausgibt, soll mir das sofort auffallen.

Zusätzlich sollen kryptographische Verfahren auch noch eine *Verbindlichkeit* her-

stellen. Wenn ich eine E-Mail von meiner Tante erhalte, dann soll meine Tante nicht abstreiten können, dass sie sie geschrieben und gesendet hat. [12]

1.5 Arten von Kryptosystemen

Grundsätzlich teilt man Kryptosysteme in zwei große Lager ein. Das erste Lager sind die *symmetrischen* Algorithmen. Bei symmetrischen Algorithmen wird für die Ver- und Entschlüsselung derselbe Key verwendet, daher auch der Name. Wenn zwei Parteien mit Hilfe eines solchen Verschlüsselungsverfahrens kommunizieren wollen, müssen sie sich zunächst auf einen gemeinsamen Schlüssel einigen. Die Sicherheit von symmetrischen Algorithmen steht und fällt somit mit der Geheimhaltung dieses Schlüssels. Wenn er in falsche Hände gerät, kann unter Umständen jeder die Nachrichten abfangen und ver- und entschlüsseln.

Symmetrische Algorithmen können wiederum in zwei Gruppen unterteilt werden. Die eine Gruppe sind die *Stream* Ciphers. Hier wird der Klartext Bit für Bit bzw. Zeichen für Zeichen verschlüsselt. Die zweite Gruppe sind die *Block* Ciphers, wo der Klartext blockweise verschlüsselt wird. Zum Beispiel in Blöcke zu je 64 Bit.

Das zweite Lager sind die *asymmetrischen* Algorithmen. Hier werden für die Ver- und Entschlüsselung zwei verschiedene Schlüssel verwendet. Der Schlüssel zur Verschlüsselung heißt Public Key, da dieser Schlüssel öffentlich gemacht werden kann oder soll. Der Sinn ist, dass wenn jemand mir eine verschlüsselte Nachricht zukommen lassen will, dieser jemand meinen Public Key nimmt, die Nachricht damit verschlüsselt und an mich sendet. Da nur ich den dazugehörigen Private Key habe, bin ich der einzige, der sie entschlüsseln kann. [12]. Das heißt allerdings nicht, dass ich auf ein persönliches Treffen unbedingt verzichten kann. Denn jemand der meinen Public Key herunterlädt und die Nachricht damit verschlüsselt, kann sich nicht sicher sein, dass es sich tatsächlich um *meinen* Public Key handelt. Ein Angreifer hätte ihm einen gefälschten unterjubeln und somit die Nachrichten abfangen und lesen können. Im weiteren Verlauf kann er dann die Nachricht verändern und mir diese Fälschung zukommen lassen. Ich denke dann, dass die Nachricht vom ursprünglichen Sender stammt und der ursprüngliche Sender glaubt, dass ich sie auf korrektem Wege erhalten habe, dabei ist zwischen uns, quasi in der Mitte, ein Angreifer, von dem wir beide nicht wissen. Dieser Angriff heißt daher auch „Man in the Middle Attack“. Obwohl ein persönliches Treffen, bzw. eine sichere Übermittlung des öffentlichen Schlüssels nicht unbedingt ersetzbar ist, bieten Public Key-Verfahren den Vorteil, dass jeder nur den eigenen Schlüssel geheim halten muss und nicht für jeden Kommunikationspart-

ner einen anderen, bei dem er sich zudem noch darauf verlassen muss, dass ihn sein Partner ebenfalls geheim hält.

1.6 Kryptographie in der Schule

Ist das Kapitel der Kryptographie für den Informatikunterricht an Schulen geeignet? Hat es Sinn, dass Kindern und Jugendlichen beigebracht wird, wie diverse Algorithmen funktionieren, warum sie vertrauenswürdig sind, wie man geheim Kommuniziert oder speichert? Die Fragen sind berechtigt. Zum einen ist Kryptographie mit Sicherheit nicht das lebhafteste Thema, das man sich vorstellen kann, viele würden sogar sagen, es ist trocken und viel zu mathematisch. Außerdem ist Kryptographie ein sehr großes und komplexes Wissensgebiet. Warum soll man die Schülerinnen und Schüler also damit behelligen, wenn man in derselben Zeit auch lernen könnte, wie man Serienbriefe mit MS Word macht?

Wie bereits eingangs kurz erwähnt, hat die Kryptographie, ob wir es wollen und merken oder nicht, schon längst Einzug in unseren Alltag gehalten. Unsere Telefonate mit dem Handy sind zumindest bis zum Funkmasten verschlüsselt, wenn wir Online auf unser Bankkonto zugreifen und Transaktionen starten wird die Verbindung zum Bankserver verschlüsselt, wenn wir beim Bankomat Geld abheben laufen im Hintergrund kryptographische Verfahren und so weiter. Natürlich *muss* man um eine Technik anzuwenden nicht wissen, wie sie im Detail funktioniert, einen groben Überblick und Grundverständnis sollte man allerdings dennoch haben.

Die zweite Begründung, warum Menschen die Fähigkeit haben sollten, ihre Daten und Kommunikation kryptographisch zu sichern, lässt manche mit den Augen rollen und hat immer einen leichten Beigeschmack von Verschwörungstheorie. Es ist sinnvoll, die eigene Kommunikation zu verschlüsseln, auch wenn man nichts zu verbergen hat (welches das Standardargument gegen Geheimhaltung ist). Der Punkt ist folgender: Man kann nie wissen, wann etwas doch lieber hätte verborgen werden sollen. Es ist nicht möglich voraus zu sehen, wie die politische Situation in einem Land in fünf oder zehn Jahren aussieht und wenn bis dahin große Mengen an Kommunikation mit protokolliert und analysiert wurde, könnte man je nach politischer, religiöser, weltanschaulicher oder sonstiger Überzeugung in Bedrängnis geraten. Bei der Volkszählung 1925 im Deutschen Reich hätte wahrscheinlich auch niemand geglaubt, dass es ein paar Jahre später verhängnisvoll sein wird, wenn man unter Religionszugehörigkeit „jüdisch“ angekreuzt hatte.

Weiters verlangt eine Demokratie uneingeschränkten und unbelasteten Diskurs über

Politik und Weltgeschehen. Es muss zur Entscheidungsfindung für die eigene Stimme möglich sein, ohne Konsequenzen im Dialog mit Mitmenschen eine Meinung zu vertreten. Das heißt aber auch, dass es nicht passieren soll, dass wenn jemand heute eine bestimmte Meinung vertritt und diese auch über elektronischem Wege kommuniziert, in Zukunft ihm das zum Verhängnis werden kann. Allein die Angst, dass das der Fall sein *könnte* verhindert die Expression und Formulierung, die man eigentlich angestrebt hat, ganz automatisch. Es ist, als ob man sich im Büro über den Chef auslässt und nicht genau weiß, ob einen der Kollege nicht hinterrücks anschwärzt, oder ob der Chef hinter der Ecke steht und mithört. Die Wortwahl wird anders ausfallen, die eigene Meinung wird zumindest nach außen hin, eine andere sein. In wunderbarer Form hat Bruce Schneier alle Argumente für Kryptographie im privaten Bereich jenseits von „weil es cool ist“ im Onlineartikel „The Eternal Value of Privacy“ für Wired.com behandelt. [13]

1.7 Vertrauen in kryptographische Systeme

Angenommen ich hätte eine Firma und diese Firma hat natürlich am freien Markt Konkurrenz. Meine Konkurrenten wollen gerne diverse Daten, die meine Firma gesammelt hat, haben. Kundendaten zum Beispiel. Natürlich will ich verhindern, dass meine Konkurrenten an diese Daten kommen. Zum einen, weil ich gesetzlich dazu verpflichtet bin und zum anderen, weil es mir geschäftlich schaden würde, wenn meine Kunden von der Konkurrenz umgarnt werden. Wenn diese Daten auf eine CD oder DVD gebrannt jetzt von Wien nach Berlin müssen, wie kann ich verhindern, dass sie in falsche Hände geraten?

Verhindern, das heißt zu hundert Prozent ausschließen, kann man diesen Fall naturgemäß nicht. Man kann die Wahrscheinlichkeit eines Datenverlustes allenfalls sehr niedrig halten.

Die Daten brennen und per Post versenden, ohne weitere Sicherheitsmaßnahmen ist die einfachste, aber auch die unsicherste Methode. Natürlich ist die Post vertrauenswürdig, wir nehmen also nicht an, dass sie den Brief öffnet und den Inhalt der CD ausliest. Aber ist der Mitarbeiter, der die CD zur Post bringen soll vertrauenswürdig? Hier steckt man in einem ähnlichen Dilemma wie der Kommandant der dem Boten nicht voll und ganz vertrauen kann. Wer garantiert, dass mein Brief nicht vom Post-LKW fällt und sie von jemandem gefunden wird, der die Daten an jemand drittes weiterverkauft? Etwas Ähnliches ist der Firma Alcatel-Lucent passiert. Ein Datenträger mit tausenden Mitarbeiterdaten ging verloren, oder wurde gestohlen. Alcatel-Lucent

kostete diese Leichtsinnigkeit viel Geld und Ansehen. [4]

Verschlüsselt ist also besser als unverschlüsselt, soviel steht fest. Aber wie soll man verschlüsseln? Welchen Algorithmus soll man wählen? Wie groß soll der Schlüssel sein? Welches Programm soll man verwenden? All diese Fragen drehen sich um einen zentralen Begriff: Vertrauen.

Es gibt eine Vielzahl an Algorithmen und es gibt zu jedem mehrere Programme, die man verwenden kann. Welchem Algorithmus kann man vertrauen und welchen Programmen? Das Dilemma ist folgendes: Wie soll ich, der Firmenchef, überprüfen können, welcher Algorithmus sicher ist und welcher nicht? Und wenn ich mich dann für einen entschieden habe, wie soll ich überprüfen können, ob das Programm, das ich verwende, auch wirklich diesen Algorithmus anwendet und nicht einen anderen der eine Hintertür hat, bei dem also ein Angreifer dann ganz leicht den ursprünglichen Klartext, in unserem Fall die Kundendaten, auslesen kann?

Bleiben wir zunächst bei der Qual der Wahl was den Algorithmus anbelangt. Da aufgrund der großen Komplexität der Überprüfung der Funktionalität eines Algorithmus ein eigenständiges Überprüfen ausgeschlossen werden kann, müssen wir uns auf jemanden verlassen, der das für uns übernimmt. Besser noch, nicht auf *jemanden* sonder auf *etwas*. Dieses Etwas ist ein Prozess an dem möglichst viele Menschen teilnehmen sollen und es ist die beste Möglichkeit zu überprüfen, ob ein Algorithmus sicher ist, oder nicht. Man greift ihn an.

Wie funktioniert das? Gescheiter Mensch A entwickelt eine kryptographische Methode. Was er jetzt nicht macht, ist ein Programm zu schreiben und dieses als das „unknackbare Super-Krypto-Programm“ zu verkaufen, sondern, er wird den Algorithmus offenlegen und seinen Kollegen rund um den Globus zugänglich machen. Wenn keiner dieser Kollegen es schafft, eine Schwachstelle in der neuen Methode von Mensch A zu finden, dann kann man sagen, dass der Algorithmus ziemlich sicher ist. Wenn es die gescheitesten Menschen auf dem Gebiet nicht schaffen, dann schafft es niemand.

Natürlich ist es möglich, dass zum Beispiel der KGB eine Schwachstelle im Algorithmus des CIA gefunden hat, dies aber selbstverständlich nicht öffentlich macht, da ja dann das CIA die Schwachstelle beheben oder einen andere Algorithmus verwenden, wird und das KGB somit nicht mehr in der Lage ist, die Nachrichten der einzelnen Agenten mitzulesen. Das ist möglich. . . Aber könnte es nicht auch sein, dass das CIA wissentlich diese Lücke lies, sich bewusst, dass das KGB die Nachrichten mitliest um sie auf falsche Fährten zu locken? Es ist also *nicht* sonnenklar, dass das KGB das größte Interesse daran hat, diese Lücke für immer für sich zu behalten. Wenn sie nämlich

diese Lücke offen legt und das CIA den Algorithmus immer noch weiterverwendet, dann weiß das KGB, dass zumindest alle Nachrichten *nach* der Veröffentlichung der Schwachstelle platziert sind und wahrscheinlich auch alle davor.

Außerdem ist der Algorithmus ja der im Idealfall der gesamten Weltbevölkerung zugänglich, das heißt, wenn er eine Lücke hat, wird diese wahrscheinlich auch von mehreren, voneinander unabhängigen Parteien entdeckt und dass alle ein Interesse daran haben, die Lücke geheim zu halten, ist eher unwahrscheinlich. Diesem Prinzip, dass möglichst viele Menschen eine Behauptung, in diesem Fall „dieser Algorithmus ist sicher“, überprüfen und ihre Ergebnisse wiederum einer Überprüfung aussetzen nennt man *Peer-Review* und ist ein entscheidender Eckpfeiler in der modernen Wissenschaft. Wenn man dem Peer-Review-Prozess also sein Leben anvertraut, indem man sich in eine Flugzeug setzt das aufgrund von peer-reviewten und angewandten physikalischen Prozessen fliegt, kann man auch einem Algorithmus, der die selbe Prüfung hinter sich hat, vertrauen.

In diesem Zusammenhang muss auch das Kerckhoffs'sche Prinzip Erwähnung finden. Dieses Prinzip besagt, dass die Sicherheit eines kryptographischen Verfahrens nicht auf der Geheimhaltung des Verfahrens, sondern auf der Geheimhaltung des Schlüssels basiert. [5]

Gut, wir wissen jetzt, dass wenn wir uns einen Algorithmus aussuchen, der die Mühlen des Peer-Review-Prozesses überlebt hat. Und welches Programm nehmen wir? Gibt es so etwas wie einen Peer-Review-Prozess für Programme? Ja, gibt es! Aber nicht alle Programme, die Menschen im täglichen Leben verwenden, werden diesem Prozess ausgesetzt. Tatsächlich ist es sogar so, dass ungefähr 90% der Heim-PC-Anwender im Jahr 2009 ein bestimmtes, sehr essentielles Programm verwendet, das sich noch nie diesem Prozess gestellt hat, nämlich das Betriebssystem Windows der US-amerikanischen Firma Microsoft. Aber wie kann sich ein Programm, das Millionen von Menschen haben des Reviews widersetzen? Um das zu verstehen, muss man wissen, wie Programme entstehen. Deshalb folgt nun ein kurzer Einschub.

Einschub zur Entstehung und Funktionsweise von Computerprogrammen Programme entstehen so ähnlich wie Kochrezepte. Beide müssen aufgeschrieben werden und dieses Aufschreiben muss widerspiegeln, wann welche Aktion wie lange und mit welchen Parametern ausgeführt wird. Die Formulierung muss in gewissen Schlüsselwörtern erfolgen und einer bestimmten Syntax entsprechen. Ein Beispiel: „Nehmen Sie 300g Zucker und rühren sie ihn den Teig bis dieser anfängt sämig zu werden.“

Hier haben wir die Anweisung eine bestimmte Menge einer bestimmten Substanz für eine bestimmte Zeit einzurühren. Der Bäcker der dies liest kann mit allen Wörtern und ihrer Konstellation etwas anfangen. Tauscht man das Wort „Zucker“ durch „Teig“ oder umgekehrt, stimmen die Wörter zwar noch, die Konstellation ist aber falsch, das Rezept ist fehlerhaft. Tauscht man das Wort „rühren“ durch das Wort „tanzen“ aus, so stimmt die Konstellation (Syntax), das Rezept ist aber trotzdem falsch, es wird nie ein Kuchen herauskommen.

So ähnlich ist es bei Programmen auch. Die zu verwendenden Wörter sind zwar etwas abstrakter und auch die Syntax ist gewöhnungsbedürftig, aber im Grunde macht man genau dasselbe, nur eben, dass der, der es ausführt nicht bäckt, sondern Variablen hoch zählt bis sie einen gewissen Wert erreichen, Wörter auf Gleichheit überprüft und so weiter.

Wir haben also einen Text mit Anweisungen und jemanden, der diese Anweisungen ausführt. Und hier sind wir an der Grenze der Kochrezeptanalogie angelangt, denn um ehrlich zu sein, führt der Computer das Programm nicht so aus, wie wir es aufgeschrieben haben. Die Wörter müssen nämlich zunächst übersetzt werden. Nachdem sie aufgeschrieben wurden, ist das Programm noch im sogenannten „Source Code“ oder „Quellcode“. Damit kann ein Computer zunächst nichts anfangen. Erst muss ein anderes Programm, ein Compiler, auf diesen Source Code angewendet werden, der den für (des Programmierens fähige) Menschen lesbaren Text in für den Computer verständlichen Binärcode umwandelt. Das heißt, eine lange Kette von Nullern und Einsern, die logisch immer noch das selbe bedeutet wie der ursprüngliche Source Code, jetzt aber in einer anderen Form vorliegt.

Wenn man wissen will, wie ein Programm im Detail funktioniert, benötigt man den Source Code, denn aus dem Studium der langen Folge von Nullern und Einsern lässt sich ungefähr so angenehm und einfach extrahieren, was das Programm macht, wie aus dem Verzehr des Kuchens herausfindbar ist, wie viel Gramm Zucker eingerührt wurden.

Wenn man Wert darauf legt, dass die Programme die man verwendet sich diesem Peer-Review-Prozess stellen, dann sollte man Ausschau nach sogenannter *freier* Software halten. Was ist das? Die gemeinnützige Organisation *Free Software Foundation*¹ hat eine Definition herausgebracht, an der man messen kann, ob Software frei ist. Es geht um vier essentielle Freiheiten, die der Autor Kraft seines Urheberrechts an

¹<http://www.fsf.org/>

der Software gewährt oder sogar garantiert oder eben einschränkt. Sobald eine der vier Freiheiten verletzt ist, handelt es sich nicht mehr um Freie Software. Diese vier Freiheiten sind [16]:

1. Das Programm für jeden Zweck zu verwenden.
2. Zu studieren, wie das Programm funktioniert und Veränderungen vorzunehmen, damit es das macht, was man will. Hierfür benötigt man den Source Code.
3. Unveränderte Kopien des Programms verteilen zu dürfen.
4. Verbesserte bzw. veränderte Versionen des Programms verteilen zu dürfen. Hierfür benötigt man den Source Code ebenfalls.

Wenn die Software, die man verwendet, die vier Freiheiten mit sich bringt, dann ist sie frei und sie entzieht sich dem Peer-Review-Prozess nicht. Manche Menschen argumentieren, dass Freiheit zwei ausreicht, um nachvollziehen zu können, was die Software macht und *prinzipiell* stimmt das. Tatsache ist allerdings, dass Software, vor allem moderne, so komplex ist, dass eine einzelne Person niemals überprüfen kann, was die Programme die sie verwendet, machen. Dafür braucht man immer eine Gemeinschaft und diese kann erst dann kooperieren, wenn alle vier Freiheiten erfüllt sind. Außerdem kann man wohl kaum von einem Peer-Review-Prozess sprechen, nur weil man selbst nachschauen darf, was das Programm macht.

2 Asymmetrische Kryptographieverfahren

Asymmetrische Kryptographieverfahren, auch Public-Key Kryptographie genannt, sind kryptographische Verfahren, bei denen der Schlüssel zum *Verschlüsseln* ein anderer ist, wie der zum *Entschlüsseln*. Der Schlüssel, der zur Verschlüsselung verwendet werden soll, wird öffentlich gemacht, daher auch Public Key, der Schlüssel zum entschlüsseln bleibt das Geheimnis des Besitzers. Er wird auch Private Key genannt. Wenn also zum Beispiel Alice und Bob sicher miteinander kommunizieren wollen, dann generieren beide für sich ein Schlüsselpaar und schicken dem jeweils anderen den Public Key. Will Alice Bob eine Nachricht übermitteln, verschlüsselt sie sie mit Bobs Public Key und kann sie nun über einen potentiell unsicheren Kanal, zum Beispiel das Internet, übermitteln. Bob entschlüsselt dann die Nachricht mit seinem Private Key. Der Vorteil ist, dass sich Alice und Bob nicht auf einen gemeinsamen Schlüssel einigen mussten, wie sie dies bei einem symmetrischen Verfahren hätten tun müssen. Bei einem symmetrischen Verfahren hätten sie sich zumindest einmal treffen müssen um sich auf einen Schlüssel zu einigen oder sie hätten einen sicheren Kanal zur Übermittlung des Schlüssels verwenden müssen, was aber manchmal unmöglich, meistens aber teuer und langsam ist (z.B. per Post).

Was der Public und was der Private Key ist, hängt vom jeweiligen Verfahren ab, das angewendet wird.

Das Verfahren muss so beschaffen sein, dass es einfach ist, mit dem Public Key zu verschlüsseln, schwer ohne den Private Key zu entschlüsseln und ebenso schwer aus dem Public Key den Private Key zu erhalten. Bruce Schneier vergleicht in seinem Buch „Applied Cryptography“ [12] Public Key Kryptographie mit dem Prinzip eines Briefkastens: Jeder kann einen Brief in meinen Briefkasten werfen. Jedoch nur der Eigentümer des Schlüssels zum Briefkasten, kann ihn öffnen und die Briefe herausnehmen. Den Briefkasten aufzubrechen hingegen ist schwer.

Zur Anwendung kommen in Public Key Verfahren Einwegfunktionen. Eine Einwegfunktion ist eine Funktion, die relativ leicht zu berechnen ist, die Umkehrung allerdings ist aufwendig. Zur Anwendung in kryptographischen Verfahren wäre dies allerdings noch nicht ausreichend. Hier braucht man eine spezielle Art der Einwegfunktion, eine Falltür-Einwegfunktion. Das Problem ist nämlich folgendes: Wenn die Verschlüsselung recht einfach geht, die Umkehrung aber kompliziert und aufwendig ist, hat die Verschlüsselung von vornherein keinen Sinn gemacht. Denn dann bräuhete derjenige, der berechtigter Weise entschlüsseln will genau so lang, wie jeder andere

Mensch auf der Welt. Hier kommt die Falltür ins Spiel. Funktionen, die eine solche Falltür haben, haben nämlich die Eigenschaft, dass die Umkehrung schwer ist, außer, man hat eine bestimmte Zusatzinformation, dann ist die Umkehrung genau so einfach, wie die ursprüngliche Funktion. Diese Zusatzinformation ist natürlich der Schlüssel, in diesem Fall, der private Schlüssel.

2.1 Use Cases

Wer sich im Internet bewegt, verwendet (hoffentlich) ziemlich oft Public Key Kryptosysteme. Zum Beispiel, wenn jemand seine E-Mails abrufen, dann logt sich dieser jemand entweder über einen Webmail-Client auf dem Server ein, auf dem seine E-Mails gespeichert sind, oder er lädt sie auf seinen Rechner herunter. In beiden Fällen muss er einen Benutzernamen und ein Passwort eingeben um sich zu authentifizieren. Aber wie kann man sicher stellen, dass das Passwort nicht von jemanden abgefangen wird? Indem man die Authentifizierungsinformationen mit dem öffentlichen Schlüssel des Servers verschlüsselt, vereinfacht gesprochen.

Eine weitere Anwendung von asymmetrischer Kryptographie ist, die vor Abhören sichere Kommunikation über unsichere Leitungen. Genauso werden häufig kryptographische Verfahren dieser Gattung dazu benutzt, Nachrichten zu signieren, um ihre Integrität, Authentizität und Verbindlichkeit zu gewährleisten. Eine Signatur ist somit nicht dazu da, den Inhalt einer Botschaft zu verschleiern, sondern eine Garantie zu geben, dass wenn die Signatur korrekt ist, ihr Inhalt in der Ausprägung ist, wie es der Urheber vorgesehen hat.

2.2 RSA

RSA ist einer der bekanntesten und weit verbreitetsten Algorithmen überhaupt. Er ist benannt nach seinen Entwicklern Ron Rivest, Adi Shamir und Leonard Adleman, welche ihn 1977 veröffentlichten. 1983 wurde er zum Patent angemeldet, welches 2000 auslief. Seine Sicherheit wird ihm von der Schwierigkeit große Zahlen zu faktorisieren, verliehen.

2.2.1 Schlüsselerzeugung, Ver- und Entschlüsselung

Die folgende Beschreibung, wie RSA funktioniert, ist der Veröffentlichung von 1977 [11] entnommen.

Um eine Nachricht M mit dem öffentlichen Schlüssel (e, n) zu verschlüsseln, macht man folgendes. e und n ist ein Paar aus positiven Ganzzahlen.

Zunächst muss die Nachricht als eine Ganzzahl zwischen 0 und $n - 1$ repräsentiert werden. Eine zu lange Nachricht muss in Blöcke geeigneter Länge gebrochen werden. Um dies zu gewährleisten, kann man die Nachricht zum Beispiel in den ASCII-Zeichensatz umwandeln und die numerischen Werte anschreiben. Die Nachricht wird dann verschlüsselt, indem man die e -te Potenz von M modulo n nimmt. Also

$$C \equiv E(M) \equiv M^e \pmod{n} \quad (1)$$

$$D(C) \equiv C^d \pmod{n} \quad (2)$$

wobei C der Ciphertext ist, E die Verschlüsselung (*encryption*) und D die Entschlüsselung (*decryption*). „Modulo“, oft auch „mod“ oder in Programmiersprachen `%`, ist die mathematische Operation, die den Rest einer Ganzzahldivision angibt. Zum Beispiel ist $10 \bmod 3 = 1$, da $10 : 3 = 3, 1 \text{ Rest}$.

So wie (e, n) der *Verschlüsselungs-Key* ist, ist (d, n) der *Entschlüsselungs-Key*.

Wie kommt man nun zu den entsprechenden Keys? Zuerst muss n berechnet werden. n ist das Produkt zweier sehr großer Primzahlen p und q . Obwohl man n veröffentlicht, bleiben p und q geheim, da es sehr schwer ist n zu faktorisieren.

Dann wird d so gewählt, dass es eine große, zufällige Ganzzahl ist, die relativ prim zu $\phi(n) = (p - 1) \cdot (q - 1)$ ist. d muss

$$\text{ggT}(d, (p - 1) \cdot (q - 1)) = 1 \quad (3)$$

wobei *ggT* für „größter gemeinsamer Teiler“ steht, erfüllen.

Die Ganzzahl e wird schließlich aus p, q und d als das Multiplikativ Inverse zu $d \bmod (p - 1) \cdot (q - 1)$ berechnet. Somit haben wir

$$e \cdot d \equiv 1 \pmod{(p - 1) \cdot (q - 1)} \quad (4)$$

Ein Beispiel:

1. $p = 47$ und $q = 59$
2. $n = p \cdot q = 2773$
3. $\phi(n) = \phi(2773) = (p - 1)(q - 1) = 2668$

4. d muss relativ prim zu $\phi(n)$ sein. Jede Primzahl die größer ist als $\max(p, q)$ ist in Ordnung. In unserem Beispiel können wir $d = 157$ wählen.
5. Um e aus d und $\phi(n)$ zu berechnen, kann man folgende Variation des Euklidischen Algorithmus anwenden: Zunächst muss der $ggT(\phi(n), d)$ gefunden werden. Berechne die Reihe x_0, x_1, x_2, \dots , wobei $x_0 \equiv \phi(n), x_1 = d$ und $x_{i+1} \equiv x_{i-1} \pmod{x_i}$ bis x_k gleich 0 gefunden wird. Dann $ggT(x_0, x_1) = x_{k-1}$. Berechne für jedes x_i die Zahlen a_i und b_i , sodass $x_i = a_i \cdot x_0 + b_i \cdot x_1$. Wenn $x_{k-1} = 1$, dann ist b_{k-1} das Multiplikative Inverse zu $x_1 \pmod{x_0}$.

In diesem Beispiel ist e somit 17.

Will man also zum Beispiel die Nachricht „STEPHANSPL_ZWOELF_H“ verschlüsseln, muss man zunächst jedem Zeichen eine Zahl zuordnen. Wichtig ist, dass beide Kommunikationspartner dies auf die selbe Art und Weise tun. In diesem Beispiel wird folgende Substitution verwendet: Unterstrich = 00, A = 01, B = 02, ..., Z = 26. Außerdem muss man die Nachricht in Blöcke geeigneter Größe unterteilen. Sinnvoll hier ist, sie in Blöcke zu je zwei Zeichen anzuordnen, somit ist die größte Zahl 2626 (ZZ) und die kleinste 0 (0000 = __). Die Nachricht wird also zu
1920 0516 0801 1419 1612 0026 2315 0512 0600 08.

Nun muss jede Zahl verschlüsselt werden. „ST“ ist somit unser erstes $M = 1920$, also

$$M^{17} = 2109 \pmod{2773}$$

So geht man nun Block für Block durch. Die Umkehrung:

$$2109^{157} \equiv 1920 \pmod{2773}$$

2.2.2 Software

Eine der bekannteste Software die RSA verwendet, ist PGP² bzw. das freie Pendant GPG³. PGP steht für Pretty Good Privacy, GPG steht für GNU Privacy Guard.

Je nach dem, welches Betriebssystem man verwendet bzw. welches Frontend für die PGP-Software, funktioniert das Erzeugen von Schlüsselpaaren, das Exportieren des öffentlichen Schlüssels, Ver- und Entschlüsseln etc. anders. Was man aber unabhängig

²<http://www.pgp.com/>

³<http://www.gnupg.org/>

von der verwendeten Software beachten sollte, ist folgendes: Das Key-Paar sollte ein Ablaufdatum haben. Es empfiehlt sich die Keys zwischen zwei und vier Jahre lang zu verwenden. Ein Grund dafür ist, dass theoretisch mit der ersten Veröffentlichung des Public Keys bzw. der ersten Übermittlung einer Nachricht schon begonnen werden kann, den Key zu knacken. Um sicher zu gehen, dass aktuelle Nachrichten nicht ausgeforscht werden können, sollte man daher immer einen Schlüssel verwenden, der im Worst Case-Szenario immer noch nicht geknackt werden könnte. Aus diesem Grund erhält man auch in regelmäßigen Abständen eine neue Bankomatkarte.

Ein weiterer Grund dafür, das Schlüsselpaar ablaufen zu lassen ist, dass Keys auch unbrauchbar werden können. Wenn man sich zum Beispiel nicht mehr sicher sein kann, ob der Private Key wirklich „private“ ist, sollte man aufhören, diesen zu verwenden. Das kann der Fall sein, wenn man merkt, dass der eigene Computer erfolgreich angegriffen wurde, etwa durch schädliche Software oder direkt durch einen Experten. Für diesen Fall lässt sich ein „Revoke“-Zertifikat erstellen. Mit Veröffentlichung dieses Zertifikats ruft man also diesen Schlüssel zurück, bzw. hebt seine Gültigkeit auf. Das Problem ist nur, dass Menschen, die den eigenen Public Key verwenden, unter Umständen dieses Zertifikat nicht erhalten, weil sie z.B. die Schlüssel nicht oft genug aktualisieren. Im Fall des Schlüsselpaars ohne Ablaufdatum ist die Dauer, die der kompromittierte Schlüssel verwendet wird nur dadurch begrenzt, wie gewissenhaft der Kommunikationspartner seine Software einsetzt. Da sich dies unserem Machtbereich entzieht, sollten wir diese Dauer nach oben hin begrenzen.

Nun war schon mehrmals die Rede von *veröffentlichen* des Public Keys. Was damit gemeint ist, ist noch nicht genau erklärt worden. Seinen Public Key kann man zunächst exportieren. Je nach verwendeter Software funktioniert dies anders. Mit GPG in Version 1.4.9 auf einem GNU/Linux-System geht das am besten mit dem Befehl `gpg --export -a 89957BD` wobei 89957BD die Key-ID des Keys ist, der veröffentlicht werden soll. Um herauszufinden, wie die ID des eigenen Keys lautet, kann man `gpg --list-keys` eingeben. Der Schalter `a` konvertiert den Output in ASCII-Zeichen. Wenn man diesen Output nun entweder kopiert oder ohnehin in eine Datei umgeleitet hat, lässt sich nun bequem ein Textfile auf der eigenen Homepage verlinken, welches der Public Key ist. Damit man es zumindest einmal gesehen hat, ein Public Key in dieser Form sieht in etwa so aus (hier ist der Key zu der Key-ID 89957BD):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v1.4.9 (GNU/Linux)
```

```
mQGibETd7ecRBACT8U4bbF4KrBro0SdRyIKbAsk3iLwmesDofcIDhkSg8VBKJ59C
nGAvBlXUZwq1ZEjt0NOJkMaImpU19GN0ZdvjUoRf1UNaCYXYC1VH7nmA0iB45lkR
1qVHQPGcBBhCoGZaHYxvCFduKiBFGMV8RZ653iduIOEWV5n3WLq2iOPuawCgszeH
N0rsDGF3V3G1FPAhDPFKrcsD/R3paviKUoQ3SD+Jl/96KmT5d/uDirU41m/KR4WM
2qZhcGh8ySfo0XxniMQJ6yv2eesJ7LDPPrRBJlSWfZw0evsIOdfnfjxtRxxqcPI7b
F4HiIHgFGemxzpnPb9Y8KMPmQNLfJTsGtcvmQ53ChYubCXVGR9bUmYFwYfTLmTX
wLPPA/9TpWf6dGsENJ7jXkwV1YbzgcAZAqG0Z5WtdwjZJoU/Kzn3WoF7kRdPGpd1
2KenHP4di1x4GboJw7eqnsfe/IZqtU2s4dcdI9K618So9U+xdEFq8BtJRgfhRBUP
w//bXHyhDb7UA6l7AqAbeKP80SS7X/1tUv+mV4PcKbk6K55uxLQfU3RlZmFuIEdy
dWJlciA8c3RncnViZXJAZ214LmFOPohmBBMRagAmAhsDBgsJCAcDagQVAggDBBYC
AwEChgECF4FAkjhBPMFCQeGNgwACgkQjzgiL4mVeb3/eQCgpr4k+7vuKta/p4/B
ollImxWGGrwan1fkngtch9IjCC3qg03k6m6UqZGfiEYEEBECAAYFAkTgoSYACgkQ
nnUApj80coJlMwCbBBxEwB6iALrQmbJInzubY8jQIi4An3413nf3Jt1cYT2ACKPD
dtxGy98eiEYEEBECAAYFAkhL7Q8ACgkQ1aXYJv+wpmXM+gCfdSdE4qurJXybjEZA
QUWPBCuyhBwAoJpnZTS6/aEijmumHiKM561tkGRUtCVTdGvmYW4gR3J1YmVyIDxz
dGvmlmdydWJlckBnbWfPbC5jb20+iGYEEExECACYFAknjiXYCGwMFCQeGNgwGCwkI
BwMCBBUCCAMEFgIDAQIEAQIXgAAKCRCP0CIviZV5vUyBAJ9YCRKYu3SRUE45Aiwz
tX3lTmn/qgCeJftCmE605TfQm8cMDz+r7fdMWqC5Ag0ERN3t9hAIAKpFCaWXQFeF
NoIIP0mqII+n2zy0yTrQJNxiIlyage94hmyLkFTHWHMY4g92abdY8m1bcm+/ug9R
S1145Ipl3nL6eP02zAjEQUMYcwpIn6qjk8BKHI8TkT7z85g2j91XSJIINJJB3HW7
4Zoe8Yz2Vka9TIvptbPF0PcExYnkzLQEHXmwZE2R4yTNJjXtgImSxtSGWxh0+N0J
7qhIhmNH4iuifnTdc6QJhsPvfCH6iU5jrF76psONA5Sy03rLkXxLxdFcqJgdjgWN
I34H8kne0KnUkcjnRyth4dUucJbJegBb1zSdKNmrMjWHxXGVx4ptowBdVD5R+KKv
kz6tRgkr/9sAAwUH/3FvejIYFP5JkNH78AfSRffaS+aiqdvGGAWoFg3RHx4Tw7sy
8pM8H8N78h1YhaG7yaxwuiczY9wIeaqEHhZsY1g5IRoJB5E4c2UGnEARwhQxttLu
xrbC7D/K/zv3tWJ7m5KrPaRGvU+kxoh0asv4AsK4RyxQ17nY2IZgoSs9hVtx43nQ
Nv7C018jmWslwJX31VPNw8YknYdLSMiWhSnucwoZa/7UKK59capWzTa38Y6zPGzX
qhQkBNryT1db9iosVlnndJaza7wE354p34r4MJAEpmoxeHu4RSiw0iqH7GDdf+AS
d2Fbt2Q7efpcLcVbmcbeuS+XYiSeWkpbPSJXTJqITwQYEQIADwIbDAUCScohYQUJ
B4X4iQAKCRCP0CIviZV5vb6MAJ9r6Nc4GM0d71JDZ4yAVHBWjb9RcQCgoauWD+bt
2S3iiK7XWP7CnxK0zjY=
=NG7S
-----END PGP PUBLIC KEY BLOCK-----
```

Noch besser als den Key direkt als Textdatei auf der eigenen Homepage zum Download anzubieten ist, den Key auf einen der zahlreichen PGP-Keyserver hochzuladen und dann auf den eigenen Key auf dem Keyserver zu verlinken. Keyserver sind Server, die PGP Public Keys speichern und verwalten. Bekannte Server sind zum Beispiel <http://pgp.mit.edu> des Massachusetts Institute of Technology oder <http://keyserver.ubuntu.com:11371/> der bekannten GNU/Linux-Distribution *Ubuntu*. Diese Keyserver synchronisieren ihre Datenbanken untereinander automatisch, man

muss die eigenen Keys also nur zu einem hochladen.

Ein weiterer wichtiger Aspekt bei PGP ist das gegenseitige signieren der Keys. Man steht wie so oft vor dem Problem, dass man der Übertragung des Public Keys vom Keyserver auf den eigenen Rechner nicht trauen darf. Es ist schließlich möglich, dass genau in diesem Augenblick, ein Angreifer zwischen dem eigenen Computer und dem Server sitzt und mir einen manipulierten Key zusendet. Insofern ist die Anfangs aufgestellte Behauptung, mittels Public Key-Verfahren kann man sicher über unsichere Kanäle kommunizieren, nur halb richtig. Wenn man wirklich sicher gehen will, bleibt einem auch in diesem Szenario nichts anderes übrig, als zumindest einmal über einen sicheren Weg die Schlüssel auszutauschen. PGP macht es einem da allerdings ein wenig einfacher. Man muss nämlich nicht den gesamten Key sicher austauschen, sondern nur eine relativ kurze Prüfsumme, den sogenannten Fingerprint. Der Fingerprint zu obigem Key lautet 7627 A752 061A 8E87 753D 4CCC 8F38 222F 8995 79BD. Mein Kommunikationspartner, nennen wir ihn Alfred, lädt sich zum Beispiel den Schlüssel von einem Schlüsselservers herunter und ruft mich im Anschluss an. Er erkennt meine Stimme und kann annehmen, dass mich niemand durch Androhung von Gewalt kontrolliert. Ich sage ihm den Fingerprint an, wenn er die gleiche Prüfsumme hat, war die Übertragung des Schlüssels einwandfrei. Was er jetzt tun kann ist, meinen öffentlichen Schlüssel mit seinem privaten Schlüssel zu signieren. Er bürgt damit, dass dieser Key tatsächlich mir gehört. Seine Signatur kann er dann ebenfalls auf den Keyserver hochladen. Der Vorteil ist jetzt folgender: Wenn Alfred einen Kommunikationspartner hat, der ihm vertraut, nennen wir ihn Berta, dann kann Berta auch mir vertrauen, nämlich über die transitive Beziehung B vertraut A, A vertraut S, B vertraut S. Auf diese Art und Weise lässt sich nach und nach ein dezentral organisiertes Vertrauensnetzwerk aufbauen. Wenn mehr Menschen im Umgang mit solcherart Software vertraut sind und die Sinnhaftigkeit und Notwendigkeit davon erkannt wurde, können diese Netzwerke explosionsartig größer werden, indem Arbeitskollegen, Schüler und Schülerinnen untereinander, Firmen gegenseitig etc. Schlüssel überprüfen und signieren. Ein Tipp noch: Wer PGP verwendet, kann den Fingerprint auch auf seine Visitenkarten drucken lassen.

2.2.3 Unterrichtssequenz

In einer Unterrichtssequenz zum Thema RSA bietet es sich an, zunächst das grundlegende Prinzip hinter Public Key-Kryptographie zu erklären, oder durch andere Arten des Unterrichts, z.B. Gruppenarbeiten, erarbeiten zu lassen. Je nach Typ der

Klasse, kann man auf die mathematischen Hintergründe eingehen, eventuell in einem fächerübergreifenden Unterricht mit Mathematik, oder Wahlpflichtfach Mathematik. In keiner Informatikklassse sollte allerdings die Anwendung von GnuPG bzw. einem Frontend dafür ein Problem darstellen.

Nachdem die theoretische Einführung abgeschlossen ist, kann man sich an die Verwendung von GnuPG machen. Hierfür ist es sinnvoll, dass die Schülerinnen und Schüler mit den einfachsten Befehlen für die Kommandozeile unter Unix oder unixähnlichen Systemen vertraut sind. Deshalb sollte dieses Kapitel erst durchgenommen werden, wenn bereits eine Einführung in Unix und die Shell gemacht wurde. In meinen Augen ist es immer Sinnvoll, die elementarsten Werkzeuge zur Lösung eines Problems kennenzulernen, bevor man (graphische) Frontends verwendet, da diese dazu neigen, Optionen, die der Entwickler des Frontends für zu kompliziert für den durchschnittlichen User hält, auszublenden. Außerdem hat man später meist keine Probleme und nur sehr kurze Umgewöhnungszeiten wenn man die Plattform wechselst, also z.B. von GNU/Linux auf Mac OS X, da die elementaren Werkzeuge nahezu immer identisch sind, oder sich nur marginal unterscheiden.

Schlüsselerzeugung Jeder Schüler bzw. jede Schülerin soll sich ein Schlüsselpaar erzeugen. Es ist fast sicher, dass wenn man nur diesen Auftrag gibt, die Frage kommt, wie dies zu bewerkstelligen ist. Je nachdem, wie der Unterrichtstyp aussieht, kann man nun die Antwort geben, oder die Schüler und Schülerinnen selbst diese Information erarbeiten lassen. Sollte letzteres der Fall sein, so reicht ein Verweis auf die Kommandozeilenoption `h` bzw. den Befehl `man gpg`.

Nachdem die korrekte Option gefunden wurde, `--gen-key`, können die Schülerinnen und Schüler beginnen, ein Schlüsselpaar zu erzeugen.

Listing 1: Schlüsselerzeugung

```
stefan@zero:~$ gpg --gen-key
gpg (GnuPG) 1.4.9;
      Copyright (C) 2008 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
  (1) DSA and Elgamal (default)
  (2) DSA (sign only)
```

```
(5) RSA (sign only)
Your selection?
```

Wenn man 1 wählt, kann man nun wählen, welche Keygröße man haben möchte. Wir wählen die größtmögliche Größe, 4096 Bit. Als nächstes sollen wir festlegen, wie lange der Key verwendet werden können soll. Keys können ein Ablaufdatum haben, welches man auch verwenden sollte. Da man im Worst-Case-Szenario davon ausgeht, dass ein Angriff auf den Key sofort nach erstmaliger Anwendung erfolgt, sollte man einen Key nur ein paar Jahre verwenden. Ein guter Wert ist zwei bis vier Jahre. Ein weiterer Vorteil an einem Ablaufdatum ist, dass wenn der Key kompromittiert wurde, das heißt wenn das System auf dem er liegt geknackt wurde, oder er auf eine andere Weise verloren ging, dann ist dies nach Überschreitung des Ablaufdatums unwichtig. In so einem Fall sollte man allerdings ein Widerrufszertifikat für den betroffenen Schlüssel erstellen (siehe Man-Page). Wir wählen 3y, also drei Jahre.

Listing 2: Schlüsselerzeugung

```
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 3y
```

Danach wird gefragt, ob das errechnete Ablaufdatum korrekt ist, was man im positiven Fall (meistens) mit y beantworten kann.

Da jeder Schlüssel jeweils einer Person zugeordnet werden soll, muss man nun Daten eingeben, die eine Person identifizieren. Hier sind es Name, eine Beschreibung wenn man dies möchte und eine E-Mail-Adresse, da diese weltweit eindeutig sind. Hier könnte man diejenige E-Mail-Adresse eingeben, die man am häufigsten verwendet. Später kann man auch noch weitere hinzufügen. Wenn man diesen Key dann zum Verschlüsseln und signieren von E-Mails verwendet, hat es allerdings keinen Einfluss,

ob dieser Identifier mit der verwendeten E-Mail-Adresse übereinstimmt. Es geht hier nur um die Eindeutigkeit.

Listing 3: Schlüsselerzeugung

```
Key expires at Fre 26 Okt 2012 21:15:13 CEST
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs
the user ID from the Real Name, Comment and Email Address in this
form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Stefan Gruber
Email address: stefan.gruber@diplomarbeit.invalid
Comment: Demonstration
You selected this USER-ID:
    "Stefan Gruber (Demonstration)
     <stefan.gruber@diplomarbeit.invalid>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to
perform some other action (type on the keyboard, move the mouse,
utilize the disks) during the prime generation; this gives the
random number generator a better chance to gain enough entropy.
```

Hat man die Daten eingegeben, wird man aufgefordert die Passphrase für dieses Schlüsselpaar einzugeben. Der private Schlüssel wird auf der Festplatte gespeichert. Aus Sicherheitsgründen allerdings nicht im Klartext, sondern symmetrisch verschlüsselt und der Key dafür wird aus der Passphrase ermittelt, die man nun eingibt. Sie sollte daher möglichst schwer zu knacken sein, also gut 15 Zeichen lang sein, mit Sonderzeichen und Zahlen und natürlich kein Wort im herkömmlichen Sinn und es sollten keine Muster darin enthalten sein, also z.B. eine Symmetrie oder Regelmäßigkeit wie in „abba“ oder „abab“. Man kann zur Generierung von guten Passwörtern `apg` verwenden. (Siehe Abschnitt „Passwörter“ auf Seite 85.)

Hier habe ich eine garantiert nicht vergebene E-Mail-Adresse verwendet, nämlich eine deren TLD „invalid“ ist. Für die Erzeugung des Schlüsselpaars werden, wie man

lesen kann, eine Menge an zufälligen Bytes benötigt. Man kann und sollte während dessen mit der Maus wild umherfahren und unsinniges auf der Tastatur eingeben. Kleinkinder an die Tastatur von Computern setzen ist oft nicht die beste Idee, in diesem Fall sind sie allerdings eine gute Quelle für Entropie.

Schlüsselaustausch Wenn jeder Schüler bzw. jede Schülerin ein Schlüsselpaar hat, sollen sie sich nun einen Partner suchen und mit diesem den Schlüssel austauschen. Dies soll möglichst sicher erfolgen und es soll überprüft werden, ob der Schlüssel korrekt ausgetauscht wurde. Als erstes sollen Informationen zu dem gerade angelegten Key angezeigt werden. Dafür macht man folgendes (Anmerkung: Wenn notwendig wird die Ausgabe verkürzt angezeigt, um überlange Zeilen zu vermeiden. So ist zum Beispiel die E-Mail-Adresse zu sg@d.inv abgekürzt worden. Es sollte allerdings trotzdem ersichtlich sein, welche Ausgabe man erwarten kann):

Listing 4: Schlüsselaustausch

```
stefan@zero:~$ gpg --list-keys sg@d.inv
pub      1024D/32CC8E2B 2009-10-27 [expires: 2012-10-26]
uid                               Stefan Gruber (Demonstration) <sg@d.inv>
sub      4096g/0D10DF45 2009-10-27 [expires: 2012-10-26]
```

Hier sieht man die Key-ID, 32CC8E2B, wann der Key erstellt wurde, wann er abläuft, wem er gehört und nochmal ID und Erstellungs- bzw. Ablaufdatum des Private Keys.

Je nach Unterrichtsform können auch hier wieder verschiedene Ansätze verwendet werden, wie die Schüler und Schülerinnen untereinander die Schlüssel austauschen. Man könnte mehrere Möglichkeiten besprechen und dann frei wählen lassen, nur eine Möglichkeit aufzeigen, oder den Schülerinnen und Schülern selbst überlassen, wie sie dieses Problem lösen, indem sie wieder die Hilfe nach nützlichen Optionen durchlesen. Hier soll nun einer von vielen möglichen Wegen beschrieben werden.

Als erstes soll der öffentliche Schlüssel als ASCII-Datei exportiert werden. Man verwendet also `--export` um zu exportieren und `-a` um dies im ASCII-Format zu tun. Die Ausgabe wird in eine Datei mit aussagekräftigem Namen umgeleitet.

Listing 5: Schlüsselaustausch

```
stefan@zero:~$ gpg --export -a sg@d.inv
> stefans-key.txt
```

Dieser Key soll nun per E-Mail an den Partner geschickt werden. Entweder, indem man ihn in die E-Mail direkt hinein kopiert, oder indem man die Datei als Anhang

hinzufügt.

Nachdem der Public Key empfangen und gespeichert wurde, kann ihn nun der Partner importieren (in dieser Demonstration passiert nicht viel, da der Schlüssel schon vorhanden ist):

Listing 6: Schlüsselaustausch

```
stefan@zero:~$ gpg --import stefans-key.txt
gpg: key 32CC8E2B: "Stefan Gruber (Demonstration) <sg@d.inv>"
not changed
gpg: Total number processed: 1
gpg:                unchanged: 1
```

Nun ermitteln beide den Fingerprint des Keys:

Listing 7: Schlüsselaustausch

```
stefan@zero:~$ gpg --fingerprint sgr@d.inv
pub  1024D/32CC8E2B 2009-10-27 [expires: 2012-10-26]
      Key fingerprint = C77D B0F8 4F6D E2F2 F89D  BF55 B96C 79BC
                        32CC 8E2B
uid                               Stefan Gruber (Demonstration) <sg@d.inv>
sub  4096g/0D10DF45 2009-10-27 [expires: 2012-10-26]
```

Nun kann der Sender, also die Person, dessen Public Key gerade übermittelt wird, den Fingerprint vorlesen und der Empfänger liest mit und überprüft, ob sie identisch sind. Wenn ja, kann der Empfänger den Key nun als sicher einstufen und ihn signieren. Es gibt mehrere Möglichkeiten, Schlüssel zu signieren und jede hat andere Funktionen. Hier wollen wir den Schlüssel einerseits signieren, andererseits aber auch unser Vertrauen in ihn ausdrücken, da wir ja die Korrektheit von ihm sorgfältig überprüft haben:

Listing 8: Schlüsselaustausch

```
stefan@zero:~$ gpg --edit-key sg@d.inv tsign
[...]
Please decide how far you trust this user to correctly verify
other users' keys (by looking at passports, checking fingerprints
from different sources, etc.)

  1 = I trust marginally
  2 = I trust fully
```

```
Your selection? 2
```

```
Please enter the depth of this trust signature.
```

```
A depth greater than 1 allows the key you are signing to make trust signatures on your behalf.
```

```
Your selection? 1
```

```
Please enter a domain to restrict this signature, or enter for none.
```

```
Your selection?
```

```
Are you sure that you want to sign this key with your key "Stefan Gruber <stef.gruber@gmail.com>" (899579BD)
```

```
Really sign? (y/N) y
```

Nun gibt man die Passphrase ein und hat den Schlüssel signiert. Man bleibt dann im Editier-Modus-Prompt und kann diesen durch die Eingabe von q verlassen. Die Änderung sollte man speichern.

Diese Prozedur muss man jetzt nochmal in die andere Richtung machen, und schon hat man ein Schlüsselpaar getauscht und kann sich noch dazu der Korrektheit der Schlüssel sicher sein.

Nachrichten verschlüsseln Wenn die Erzeugung, Übermittlung und Verifikation der Schlüssel abgeschlossen sind, können die Schüler und Schülerinnen anfangen, erste Nachrichten zu verschlüsseln und zu übermitteln. Auch hier führen wieder viele Wege ins sprichwörtliche Rom. Eine Möglichkeit ist, einen Mailclient wie Mozilla Thunderbird oder KMail zu verwenden. Letzteres hat GPG-Unterstützung schon eingebaut, bei Thunderbird ist sie durch ein Plugin namens *Enigmail*⁴ nachzurüsten. Aber man kann auch alles händisch machen, was keine schlechte Idee ist, um die einzelnen Schritte die erledigt werden, nachvollziehen zu können.

Im ersten Schritt braucht man zunächst eine Nachricht, die man versenden möchte. Diese kann alles sein, was man in einem Computer speichern kann, Text, Bild, Video,

⁴<http://enigmail.mozdev.org/>

vollkommen egal. In diesem Beispiel wird Text verwendet, eine kurze Passage aus Dante Alighieris „Die Göttliche Komödie“.

Listing 9: Schlüsselaustausch

```
stefan@zero:~$ cat dante.txt
Auf halbem Weg des Menschenlebens fand
ich mich in einen finstern Wald verschlagen,
Weil ich vom rechten Weg mich abgewandt.
Wie schwer ist's doch, von diesem Wald zu sagen,
Wie wild, rauh, dicht er war, voll Angst und Not;
Schon der Gedank' erneuert noch mein Zagen.
Nur wenig bitterer ist selbst der Tod;
Doch um vom Heil, das ich drin fand, zu kuenden,
Sag' ich, was sonst sich dort den Blicken bot.
```

Das Verschlüsseln geht jetzt recht einfach:

Listing 10: Schlüsselaustausch

```
stefan@zero:~$ gpg -r 0x32CC8E2B -e dante.txt
```

Das `r` steht für „Recipient“, also Empfänger. Das `e` steht für „encrypt“. Nach Abgabe dieses Kommandos wird eine verschlüsselte Datei mit Namen *Name der ursprünglichen Datei.gpg* erstellt. Diese kann man jetzt als Anhang in einer E-Mail versenden. Wenn man will, kann man sie auch umbenennen, damit man eventuell aus dem Dateinamen und der Endung nichts herauslesen kann. Oder man macht noch etwas viel cooler. Man verwendet das Programm `base64`. Base64 encodiert binäre Dateien, wie zum Beispiel Programme, Zip-Archive oder verschlüsselte Daten in ASCII-Zeichen um, die man dann zum Beispiel per E-Mail versenden kann. Dazu wendet man einfach `base64` auf die Datei an:

```
stefan@zero:~$ base64 dante.txt.gpg
hQQ0AzlezmANEN9FEBAAoN/h/q2DxRCHhVudd0N84fqddQaTDkSuexFLEBuxXxdN2YcDyWF1ZKzv
3oNXXR8DAsny1MAE9NMjA14aM+uq/8ga03LymZMC61zFAAmYAPTai2qmoTSn6hu/7fyWsWYb6XqX
XNUH4XUUATsd9/Eiz061DQbZ2AwKEjHYTMsNj18dPr+WOUEhruSP+91IOLGr01+L6EKJnHsxj bv3
/qu1mmvP95g/XfKEjX41N57Fli6G0f9INNsFlPXIHUzB6th0udCoEaIVeXz1x9rcGXsprD8cUAY8
o21YJz5YQSY/DLRncroxZXf0BvZy1o27ocEUB1i0kWSod50R3W9RXtSPtmDUey7313LcrcRti3fR
ZiKeQ0Y+3QHv10VCLMW7fo8MbCKTdLKONUdmlIOfch2WdH0XpV+1Wp504q3BKIpZXpxxLBoZz4WR
KsvPMdyjfqfKd02cSMcjod6MfAS448zAarRoc/E+xAe0ZnGJa5sbueSsZsdRaPJgB/iB2v3axE1F
fwEXUawxqQ29S0mIxrt1Y3EkVU4TA9YC9aLNIFAsqs0BawS0Juffi0wvU0pMYdy2kS1BU5ruBYP4
```

tyFDyyfldDLIh48AQ8KocMHRAQHn1CU/a8Wy3tfM02E+1Bwf93A1hqvVaJ0+B/Nsa02B92vpwE7M
CvdCf4+dRwc0o0KujGYP/jwRkcQTLyMixJiLbIyjXPeadatqlzta1d3Y5K/mEzapng1CS08+UHPU
4iq2wwUZtMetxCKiAqLCW+K+ii65BzuFL7/Vk6KuxzuHaqRukowMPI00ueR8jPMyZWUBrjHHULab
FwCVBDkMfY2b0Zk4zHzMEKKN+vcGdYQnqrQWxCwSa8idjr0FecZF/eTcadD0gPoChtcohi1driGZ
+A745s1K+xtSdw/ugLf4H0AHby3mTSHPAYk+18p79mqCYGqTcFKajPPWNxUc1lmzkHhb24TaVo4V
l+oAl+j5+1Y6/GcjkTwS5wH9nxYrxisidAhg0etD4rGNi8omj75i+Qb5zNwUJqwReVIhvc2/MjJ6
xBn9DP2rh4gvCedaGFldVDcM1WtqF9jfQ7gqJ5++1ZUaUuPEjdG3r9Xn7ak1ArRt9z/qLBgt93oo
RT2Jtihv9o1056txbH9FnJ24Eovfpd9TdvTrLfAxrNd0TjJqsC4sJtb5d7EDgx7nNFx1EKq9XSIw
Gm2Ij7rsx0inqYi4UXJ3Jo6AYJovUNxpvgCqbbiiu3vgZvxJww/f+Lrr1XQInBiB/MDfIGv0UlyA
g5qcuNEHtWdi0zYTwf800SmTKv12GqJY2CWsjfBMX/S6COPGXtnt7sDYgsuDmPI5v+dTY3KRFumB
IZy6Dxw5FudT1qTroM5N0sCMAVBN8La5AmDTT76YbuCZQCDeedCOMma/lDgI7ceUSg5skyYhv09N
yDvnq26z1wHP/4L0W3sS+W7X0E1chQFhSBDAU61eyCHG+x3YAICN1Q+nDqjgh6uoHVom7RiuuGJc
rm4ejvRauhC351z4jDM9u2b+dDxeLe8vXk2xVncocs8NI+GS2QZWeWxIpSykkY2ZypQBe+L7W0g2
+2vNgmxJV7qR56rTryIoNUqPxD62nG2dTVZGGfbkNysJqRh1N90PIHFtZTzCgk0moTiH/9AICPZw
fTDwdPeIu853z8YNifhD5WH06w7DaztZwGn62X6IsNK0SNMkyhRsN6KcWIX0SyhwNECva27Z2+XJ
YVvk9iD0TEU6+OUK+/K4exN73b7zh2xP1SgRLqx+52d1/NMFp3wR2Qv3DQCW0aubunvWQAjMIc3ak
NpryE2M+sU=

Das Ist-Gleich-Symbol am Schluss markiert das Ende der Datei. Wenn dieser Text zu lang ist, kann er auch in eine Datei umgeleitet werden. Jedenfalls wird er dann in eine E-Mail kopiert und an den Empfänger versendet. Dieser macht dann nichts anderes, als den Text in eine Datei zu kopieren, zum Beispiel „nachricht.b64“ und dann den Befehl `base64 -d nachricht.b64 > nachricht.gpg` (das d steht für „decode“) anzuwenden um anschließend die GPG-Datei zu entschlüsseln. Dies geschieht mit `gpg -d nachricht.gpg > nachricht.txt`. Und schon kann man die Nachricht mit zum Beispiel `less nachricht.txt` lesen. Sollte der Empfänger nicht wissen, welche Art von Datei verschickt wurde, dann ist `file Datei` immer recht zuverlässig zu erraten, um was es sich handelt.

2.3 DSA

DSA steht für Digital Signature Algorithm und ist ein Standard der Bundesregierung der USA. 1993 wurde der ursprünglich von einem NSA-Mitarbeiter (National Security Agency) patentierte Algorithmus in den Digital Signature Standard übernommen. Das Patent wurde an die US-Regierung übertragen und ist weltweit lizenzkostenfrei verwendbar.

2.3.1 Schlüsselerzeugung, Signieren und Überprüfen

Die Schlüsselerzeugung geschieht wie folgt:

- Zunächst muss eine Primzahl p der Länge L Bit gewählt werden. Dabei gilt, dass $512 \leq L \leq 1024$ und L ein Vielfaches von 64 sein muss.
- Nun wählt man eine weitere Primzahl q die 160 Bit lang ist und darüber hinaus ein Teiler von $p - 1$ ist.
- Als nächstes muss h gewählt werden, für das gilt, dass $1 \leq h \leq p - 1$ und $h^{\frac{p-1}{q}} \bmod p \neq 1$
- Nun berechnet man $g = h^{\text{frac}p-1q} \bmod p$
- Nun muss man x zufällig wählen, jedoch muss gelten $1 < x < q$.
- Jetzt muss $y = g^x \bmod p$ berechnet werden.

p, q, g und y sind der öffentliche Schlüssel, x ist der private.

Signieren einer Nachricht: Wir wollen die Nachricht m signieren. SHA-1(m) bezeichnet den SHA-1 (Secure Hash Algorithm) Hashwert der Nachricht m .

- Für jede zu signierende Nachricht muss ein zufälliges s gewählt werden, für das gilt $1 < s < q$.
- Nun muss $s_1 = (g^s \bmod p) \bmod q$ berechnet werden.
- Danach muss $s_2 = s^{-1} \cdot (\text{SHA} - 1(m) + s_1 \cdot x) \bmod q$ berechnet werden.

(s_1, s_2) ist die Signatur der Nachricht. s ist geheim zu halten, da mit s der private Schlüssel x berechenbar wäre.

Überprüfung einer Signatur: Wir erhalten eine Nachricht m und die Signatur dazu (s_1, s_2) .

- Als erstes wird überprüft, ob $0 < s_1 < q$ und $0 < s_2 < q$ gilt. Wenn nicht, ist die Signatur ungültig.
- Berechne: $w = s_2^{-1} \bmod q$

- $u_1 = SHA - 1(m) \cdot w \bmod q$
- $u_2 = s_1 \cdot w \bmod q$
- $v = (g^{u_1} \cdot y^{u_2} \bmod p) \bmod q$
- Wenn $v = s_1$ gilt, dann ist die Signatur gültig.

[10]

2.4 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) ist ein Verfahren zur Public Key-Verschlüsselung, welches 1985 von Neal Koblitz und Victor Miller entdeckt wurde. ECC bietet die selben Funktionen wie RSA, jedoch basiert die Sicherheit dieser Verfahren auf einem anderen Problem, dem *Elliptic Curve Discrete Logarithm Problem*. Die zur Zeit besten Algorithmen dieses Problem zu lösen benötigen exponentielle Laufzeit, während im Gegensatz dazu Algorithmen zur Faktorisierung von Ganzzahlen im subexponentiellen Laufzeitverhalten liegen. Aus diesem Grund kann man bei ECC mit deutlich kleineren Schlüsseln als bei RSA auskommen. Zum Beispiel gilt ein 160-Bit Key bei ECC als gleichsicher wie ein 1024-Bit Key bei RSA. Der Vorteil der kleineren Keygrößen sind unter anderem Geschwindigkeit, bessere Ausnutzung von Bandbreite und effizientere Speicherung, was besonders da Gebrauch findet, wo Speicher nur sehr begrenzt vorhanden ist, z.B. Smartcards.

Was man mit elliptischen Kurven in der Kryptographie somit macht ist, sich eine Umgebung erschaffen, in der es ein bestimmtes Problem gibt, das unseren vorherigen Ansprüchen einer Falltüreinwegfunktion entspricht. Eine elliptische Kurve E über \mathbb{R} ist definiert als

$$y^2 = x^3 + ax + b \tag{5}$$

wobei

$$a, b \in \mathbb{R} \text{ und } 4a^3 + 27b^2 \neq 0 \tag{6}$$

So eine Kurve sieht zum Beispiel aus wie in Abbildung 2⁵ auf Seite 32.

⁵Quelle: http://de.wikipedia.org/w/index.php?title=Datei:E11_kurve.png&filetimestamp=20041112233133

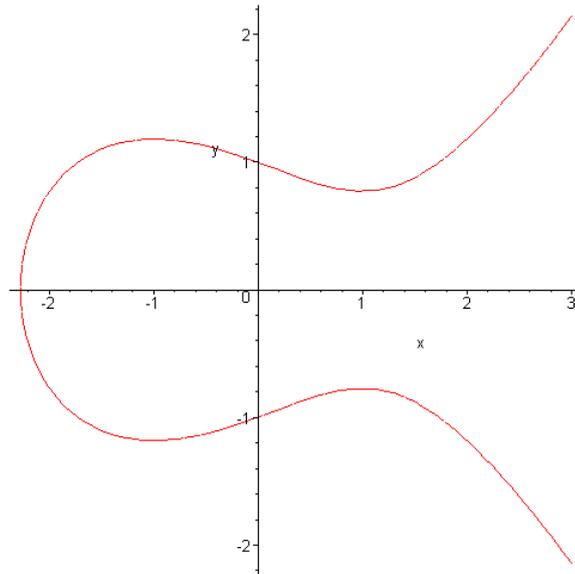


Abbildung 2: Elliptische Kurve über \mathbb{R} .

Wenn man allerdings Kurven über \mathbb{R} in der Informatik verwenden würde, stößt man früher oder später auf ein Problem. In \mathbb{R} können Zahlen unendlich viele Nachkommastellen haben und unendlich groß werden. Ein Computer kann aber nur endlich viele Zahlen speichern. Somit erhält man, wenn man nicht mit „schönen“ Zahlen arbeitet immer Rundungsfehler die das gesamte Vorhaben unmöglich machen. Deshalb betrachtet man nicht die Kurven über den gesamten \mathbb{R} , sondern über ein Feld \mathbb{F}_p von Zahlen modulo p , wobei p eine sehr große Primzahl ist. Das hört sich jetzt komplizierter an als es ist. Alles was man macht ist, nicht unendlich viele Punkte herzunehmen, aus denen die Kurve bestehen kann, sondern nur aus $p - 1$ Punkten. Betrachtet man eine solche Kurve, und ist p nicht besonders groß, dann erkennt man überhaupt nicht, dass es sich um eine Kurve handeln soll. Alles was man sieht, sind Punkte auf einem Koordinatensystem. Was man in Feldern \mathbb{F}_p noch hinzufügen muss ist in Gleichung 6 ein $(\text{mod } p)$, sodass sie so aussieht wie

$$a, b \in \mathbb{R} \text{ und } 4a^3 + 27b^2 \not\equiv 0 \pmod{p} \quad (7)$$

Zum Beispiel kann E eine elliptische Kurve über \mathbb{F}_7 und durch die Gleichung

$$y^2 = x^3 + 2x + 4 \quad (8)$$

definiert sein. Dann sind die Punkte aus denen E besteht

$$E(\mathbb{F}_7) = \{\infty, (0, 2), (0, 5), (1, 0), (2, 3), (2, 4), (3, 3), (3, 4), (6, 1), (6, 6)\} \quad (9)$$

Der Punkt ∞ ist der Punkt im Unendlichen und wird jeder Kurve hinzugefügt. Wenn durch zwei Punkte die exakt übereinander liegen eine Gerade gelegt wird, die also parallel zur y -Achse ist, dann wird der Punkt ∞ von dieser Gerade geschnitten.

2.4.1 Schlüsselerzeugung

Sei E eine elliptische Kurve über dem endlichen Feld \mathbb{F}_p und sei P ein Punkt auf $E(\mathbb{F}_p)$. Außerdem ist P von der Primzahlordnung n . Die Primzahl p , die Gleichung der elliptischen Kurve E , der Punkt P und seine Ordnung n sind die Parameter des Public Keys. Für den Private Key wird eine natürliche Zahl d , die gleichverteilt zufällig aus dem Intervall $[1, n - 1]$ gewählt wird, benötigt. Der zugehörige Private Key ist dann $Q = dP$.

2.4.2 Ver- und Entschlüsselung

Der Klartext m wird zunächst als Punkt M repräsentiert. Zu M wird dann kQ hinzu addiert, wobei k eine zufällige Ganzzahl ist und Q der Public Key des Empfängers. Der Sender übermittelt $C_1 = kP$ und $C_2 = M + kQ$. Der Empfänger verwendet seinen privaten Schlüssel d und berechnet

$$dC_1 = d(kP) = k(dP) = kQ \quad (10)$$

und kann damit $M = C_2 - kQ$ wieder herstellen.

[3]

2.4.3 Unterrichtssequenz

Die Verwendung von Programmen denen Elliptische Kurven Kryptographie zu Grunde liegt, unterscheidet sich kaum von der Verwendung jener, die andere mathematische Grundlagen ausnutzen. Weil es sich aber anbietet, möchte ich hier eine Seminararbeit von mir in Zusammenarbeit mit dem Studenten Marko Kristof einbauen, in der versucht wurde, einige grundlegende Operationen auf elliptischen Kurven mittels Geometrie zu veranschaulichen. Die Arbeit wurde in der Lehrveranstaltung „Didaktik der

Computermathematik“ im Sommersemester 2009 bei Univ.-Prof. Dr. Wilfried Grossmann und Ao. Univ.-Prof. Dr. Erich Neuwirth eingereicht und wurde mit „Sehr Gut“ benotet. Geogebra⁶ ist freie Software nach Definition der Free Software Foundation und kann zum zeichnen mathematischer Figuren verwendet werden.

Zeichnen Zunächst möchten wir eine elliptische Kurve zeichnen. Da wir uns veranschaulichen wollen, wie die Gestalt der Kurve in Abhängigkeit zu den Koeffizienten a und b (siehe Gleichung 5) aussieht, erstellen wir zwei Schieberegler für die freien Zahlen a und b . Ihr Wertebereich soll zwischen -10 und 10 liegen (Abb. 3).

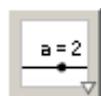


Abbildung 3: Button für Schieberegler

Da es in Geogebra nicht möglich ist, Quadrate von Funktionen einzugeben, müssen wir die vorherige Formel nach y umformen und die beiden Äste der Wurzelfunktion separat plotten. Wir geben also in die Kommandozeile am unteren Fensterrand $y = \sqrt{x^3 + a*x + b}$ um den oberen Ast zu plotten und danach $y = -\sqrt{x^3 + a*x + b}$ für den unteren. Das Ergebnis sollte in etwa wie in Abb. 4 aussehen.

Nun kann man ein wenig mit den Schiebereglern herumexperimentieren und sich überlegen, ob die Behauptung von vorhin, nämlich dass eine Gerade durch zwei Punkte immer durch genau einen weiteren Punkt geht, stimmt.

Gerade durch zwei Punkte Um zu veranschaulichen, ob die elliptische Kurve tatsächlich diese Eigenschaft hat, zeichnen wir zwei Punkte auf einem der Äste. Dafür aktivieren wir den Button zum Zeichnen von Punkten (Abb. 5(a)) und klicken zweimal auf eine Stelle der Kurve. Danach auf den Button zum Verbinden von zwei Punkten durch eine Gerade (Abb. 5(b)) und klicken zunächst den ersten Punkt und danach den zweiten an. Das Ergebnis sollte wie in Abb. 5 auf Seite 36 aussehen.

Nun kann man die Punkte verschieben um zu sehen, dass die zuvor beschriebene Eigenschaft, tatsächlich vorhanden ist. Leider haben wir keine Möglichkeit gefunden, die „beiden“ Kurven zu verbinden, sodass man die Punkte auch auf den anderen Ast ziehen kann.

⁶<http://www.geogebra.org/>

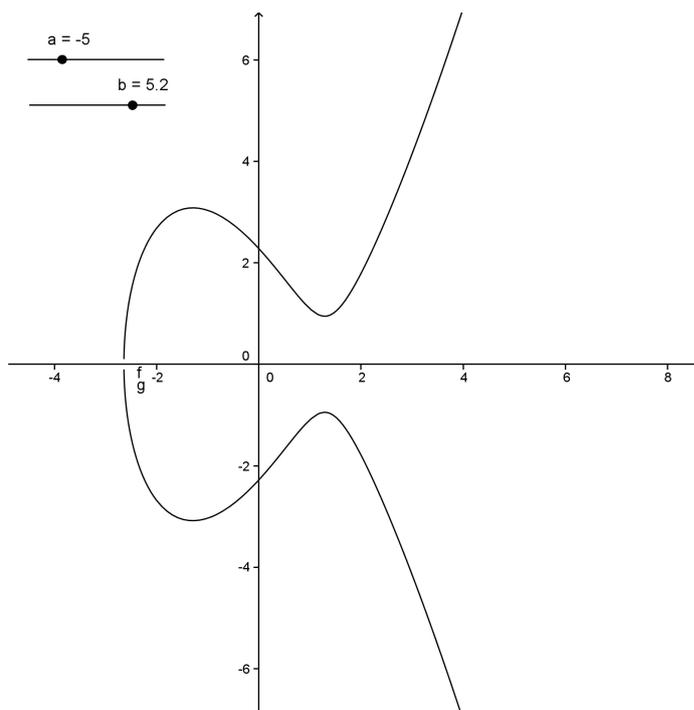


Abbildung 4: Die elliptische Kurve wird in zwei Funktionen aufgeteilt



(a) Punkte zeichnen



(b) Punkte verbinden

Addieren auf der elliptischen Kurve Im Tutorial „Krypto-Verfahren basierend auf elliptischen Kurven“⁷ kann man nachlesen, wie die Addition auf einer elliptischen Kurve definiert wird. Dies wollen wir hier grafisch mit Geogebra nachvollziehen. Dazu machen wir beim vorherigen Beispiel weiter. Zwei Punkte werden so addiert, indem man vom dritten Punkt, wir nennen ihn C , der der Schnittpunkt der Verbindungsgerade der ersten beiden und der Kurve ist, eine senkrechte Gerade zeichnet, die die Kurve abermals in genau einem Punkt schneidet. Dieser Punkt D ist das Ergebnis der Addition von A und B (siehe Abb. 6).

Wird ein Punkt verdoppelt, also z.B. $A + A$ berechnet, legt man in A die Tangente an die Kurve und zeichnet vom Schnittpunkt der Tangente aus, die senkrechte Gerade, um die Summe von A mit sich selbst zu erhalten. Sind A und B genau übereinander, so ist die Verbindungsgerade senkrecht. Diese wird die Kurve nicht mehr schneiden,

⁷Quelle: <http://www.warendorf-freckenhorst.de/elliptische-kurven/frame.html> [Online; Stand 25. April 2009]

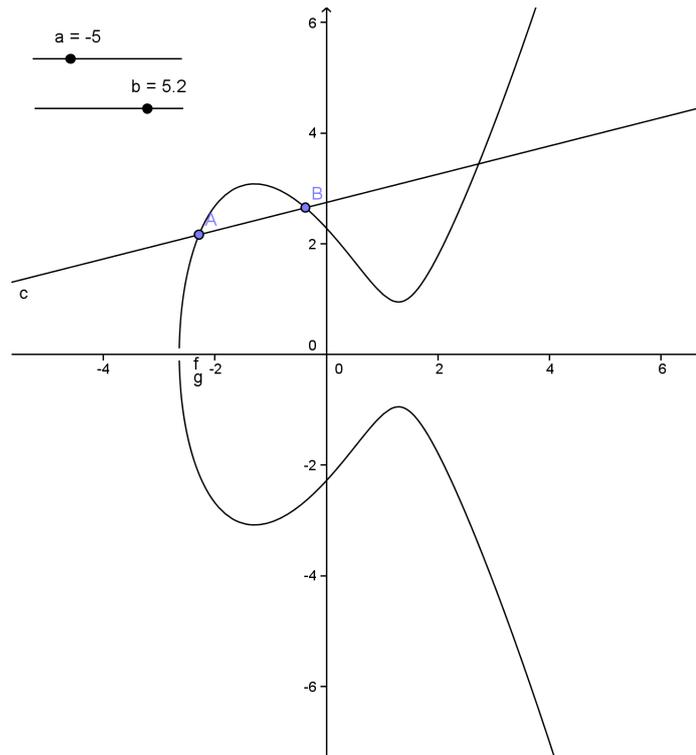


Abbildung 5: Die Gerade schneidet noch in genau einem weiteren Punkt

aber für diesen Fall wurde der Punkt ∞ definiert.

Unterrichtsplanung Das Thema asymmetrische Kryptosysteme lässt sich vielseitig sowohl im Informatikunterricht, als auch im Mathematikunterricht verwenden. Wir sehen die Veranschaulichung von elliptischen Kurven und Rechenoperationen auf ihnen als Teil einer größeren Unterrichtssequenz an, die sich mit Kryptosystemen allgemein beschäftigt. Geogebra ist zwar gut, um sich die allgemeinen Definitionen vor Augen zu führen, die für Kurven über \mathbb{R} gelten, in der Realität werden allerdings Restklassenräume definiert, um eine genaue Addition von Punkten auf der Kurve zu gewährleisten. Diese Kurven über Restklassenräume sind für die Schülerinnen und Schüler wohl kaum mehr als ein paar Punkte auf in einem Koordinatensystem, auch lassen sie sich nicht in Geogebra zeichnen. Deshalb wäre diese Fortführung der Materie nur der Vollständigkeit wegen erwähnt, würde aber nicht mehr vorgetragen, oder erarbeitet werden.

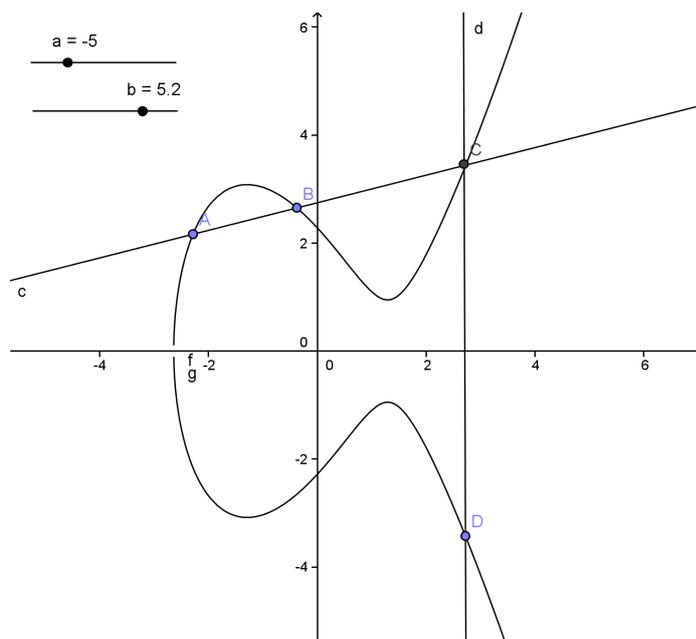


Abbildung 6: Addition von zwei Punkten die nicht identisch sind

3 Symmetrische Kryptographieverfahren

Im Gegensatz zu asymmetrischen Verschlüsselungsverfahren werden bei symmetrischen zum Ver- und Entschlüsseln der selbe Schlüssel verwendet. Das hat den Nachteil, dass der Schlüssel zwischen Sender und Empfänger sicher übertragen werden muss. Es muss zum Beispiel bei einem persönlichem Treffen ein Schlüssel ausgetauscht werden. Der Vorteil von symmetrischen Verfahren ist allerdings, dass sie meist weniger aufwendig sind und mit kürzeren Schlüsseln auskommen. Der Nachteil, dass der Schlüssel zwischen Sender und Empfänger über einen sicheren Kanal ausgetauscht werden muss fällt weg, wenn Sender und Empfänger die selbe Person sind. Das ist zum Beispiel dann der Fall, wenn man die eigenen Datenträger verschlüsselt. Dies zu tun ist ohnehin eine gute Idee, da man Festplatten besser nicht einfach so wegschmeißt oder verkauft, wenn man sie nicht mehr benötigt. Denn auch wenn man die Dateien darauf „löscht“, sind sie meistens relativ leicht wieder herzustellen. Bei gängigen Kapazitäten von rund 500 GB ist es allerdings sehr zeitaufwändig die gesamte Festplatte mit Bitmustern und Zufallszahlen zu überschreiben. Will man somit eine alte Festplatte loswerden, sollte man sie also am besten zerstören, oder eben die Daten darauf verschlüsseln.

Man kann jedoch die Vorteile von symmetrischen und asymmetrischen Verfahren gut miteinander kombinieren. Es bietet sich förmlich an, dass man einen symmetri-

schen Schlüssel, der den Vorteil hat klein zu sein, asymmetrisch verschlüsselt über einen unsicheren Kanal übermittelt. Somit fällt die Notwendigkeit eines sicheren Kanals weg und man hat trotzdem die Vorteile der symmetrischen Kryptographie, wie oben beschrieben.

Symmetrische Verschlüsselungen werden zudem in *Stream-* und *Block-Cipher-*Verfahren eingeteilt. Bei Stream-Cipher-Algorithmen wird der Klartext Zeichen für Zeichen verschlüsselt, während beim Block-Cipher Blöcke bestimmter Größen verschlüsselt werden.

3.1 DES

DES, was für *Data Encryption Standard* steht, manchmal auch DEA (Data Encryption Algorithm) genannt, ist ein in den 1970er Jahren entwickelter und veröffentlichter, blockorientierter, symmetrischer Verschlüsselungsalgorithmus, der explizit für den zivilen Gebrauch entworfen wurde. Zu dieser Zeit war Kryptographie und die Forschung in diesem Gebiet ein wenig planlos. Es gab ein paar private Unternehmen die mehr oder weniger sicheres kryptographisches Equipment verkauften. Dabei kochte jeder sein eigenes Süppchen und nichts war miteinander kompatibel. Auf westlicher Seite gab es niemanden, der sich wirklich mit der Materie auskannte, nichts und niemand, der vernünftig kryptographische Verfahren analysieren und zertifizieren konnte. Das größte Wissenskapital für solcherlei Sachen hatte das US-amerikanische NSA (National Security Agency) die aber nicht mal wirklich ihre eigene Existenz zugab, geschweige denn irgend welche von ihnen für den militärischen Bereich entwickelten Sachen preisgab.

Offensichtlich gab es einen großen Bedarf von nicht-militärischer Seite, an einem vernünftigen, standardisierten Algorithmus, den man leicht in Hard- und Software implementieren konnte. Also startete das NBS (National Bureau of Standards), das heutige NIST (National Institute of Standards and Technology) eine Ausschreibung an die Öffentlichkeit, doch bitte Algorithmen einzusenden, die ein paar wichtige Kriterien erfüllen, damit die dann überprüft und zertifiziert und als Verschlüsselungsstandard verwendet werden können. Diese Kriterien waren:

- Der Algorithmus musste ein hohes Maß an Sicherheit gewährleisten.
- Er musste vollständig spezifiziert und einfach zu verstehen sein.

- Die Sicherheit des Algorithmus muss vom Schlüssel abhängen und nicht von der Geheimhaltung des Algorithmus selbst. (Kerckhoffs Prinzip)
- Der Algorithmus muss für alle Verwender zugänglich sein.
- Er muss in diversen Anwendungen anwendbar sein.
- Implementierbar in elektronischen Geräten.
- Effizient in der Anwendung.
- Es muss möglich sein, den Algorithmus zu evaluieren.
- Der Algorithmus musste exportierbar sein.

Zwar war die Antwort von öffentlicher Seite auf diese Ausschreibung ein Indiz dafür, dass es Bedarf gab, leider erfüllte aber keiner der Einsendungen auch nur annähernd diese Kriterien.

Das NBS gab aber nicht auf und startete einen zweiten Aufruf und erhielt ein Jahr später auch einen vielversprechenden Kandidaten. Den von IBM entwickelten Algorithmus *Lucifer*.

Nachdem ein paar patentrechtliche Formalitäten aus dem Weg geräumt waren, bat das NBS das NSA um Hilfe, den Algorithmus zu evaluieren und anzupassen. Außerdem wurde wieder an die Öffentlichkeit appelliert, Kommentare zu dem Verfahren abzugeben. Diese kamen auch, reichlich. Die meisten davon drehten sich aber nicht um den Algorithmus an sich, sondern um die Sorge, dass sich das NSA so stark beteiligte. Es wurde vermutet, dass das NSA eine Hintertür einbaut, um verschlüsselte Daten auszulesen. Ein weiterer Grund zur Sorge war, dass die ursprüngliche Keygröße von 128-Bits auf 56-Bits reduziert wurde und wie der Algorithmus intern arbeitet (siehe nächster Abschnitt). Heute würde das niemanden mehr Grund zur Sorge bereiten, damals aber war es mysteriös und besorgniserregend.

Im Jahr 1976 wurde DES als Bundes-Standard angenommen und für alle unklassifizierten Kommunikationen der Regierung zugelassen. 1980 und 1981 wurden „DES Modes of Operation“ und „Guidelines for Implementing and Using the NBS Data Encryption Standard“ veröffentlicht. So etwas gab es bisher noch nie. Einen von der NSA evaluierten und als sicher geltenden Algorithmus in öffentlicher Hand. Dabei konnte diese Tatsache auf ein Missverständnis Seitens NSA zurückgeführt werden. Diese nahm nämlich an, dass der Algorithmus nur in Hardware implementiert werden sollte und nicht, dass jetzt jeder DES-Software schreiben konnte. Das NSA hat

DES als „ihren größten Fehler“ bezeichnet und hätten nie an der Zusammenarbeit zugestimmt, hätten sie gewusst, dass später Menschen diesen Algorithmus in Software schreiben und verwenden konnten. Doch dafür war es zu spät, zum Glück. Denn kaum ein anderes Ereignis hat die Entwicklung im Feld der Kryptographie so vorangetrieben. Jetzt gab es nämlich endlich einen Algorithmus zu studieren, den das NSA als sicher ansah. Der nächste vom NSA entwickelte Algorithmus „Skipjack“ war dann übrigens classified. In den 1980er Jahren stürzten sich dann alle wie wild auf DES. Vor allem Banken. [12]

Durch die kurze Schlüssellänge von nur 56-Bits ist DES seit längerem nicht mehr sicher. Dies kann man aber korrigieren, indem man ihn mehrmals hintereinander anwendet. Ein Beispiel dafür ist Triple-DES, oder auch 3DES genannt. Ansonsten ist DES konzeptionell nach wie vor sicher. [8] Bei einer Known-Plain-Text-Attacke kann der Schlüsselraum zwar halbiert werden, das bedeutet aber anders ausgedrückt nur, dass man von einem 56-Bits Key auf einen 55-Bits Key reduzieren kann. Da aber 56-Bits auch schon zu wenig sind, kann man diese Schwäche ruhig als geringfügig einstufen.

3.1.1 Funktionsweise von DES

Obwohl vorhin die Rede davon war, dass DES einen 56-Bits Key verwendet, ist es tatsächlich so, dass der Key 64 Bits groß ist. Davon sind allerdings nur 56 zufällig, die restlichen acht können zur Fehlerkorrektur verwendet werden. [8]

Der Algorithmus verschlüsselt Blöcke von 64 Bits Größe zu 64 Bits großen Ciphertext-Blöcken. Jeder Block besteht aus Bits die von links nach rechts durchnummeriert werden, wobei das am weitesten links stehende das Bit Nummer Eins ist. Zur Entschlüsselung muss der selbe Key wie zur Verschlüsselung verwendet werden (daher auch symmetrisch), wobei die Reihenfolge mit der die Key-Bits angewendet werden verändert ist. Ein Block der verschlüsselt werden soll wird zunächst einer anfänglichen Permutation IP unterzogen, danach eine komplexe, vom Schlüssel abhängige Berechnung und am Schluss einer, zur ersten *inversen*, Permutation IP^{-1} .

Die komplexe, vom Schlüssel abhängige Berechnung kann wiederum in zwei Funktionen unterteilt werden. Die Funktion f und die Funktion KS , was für Key Schedule steht. Ein Flowchart, wie die Verschlüsselung mit DES abläuft, sieht man in Abb. 7 auf Seite 41.

Die IP -Permutation funktioniert wie folgt: Die 64 Bits des Input-Blocks werden so permutiert, dass das erste Bit 58 ist, das zweite 50, dann 42 usw. bis am Ende das

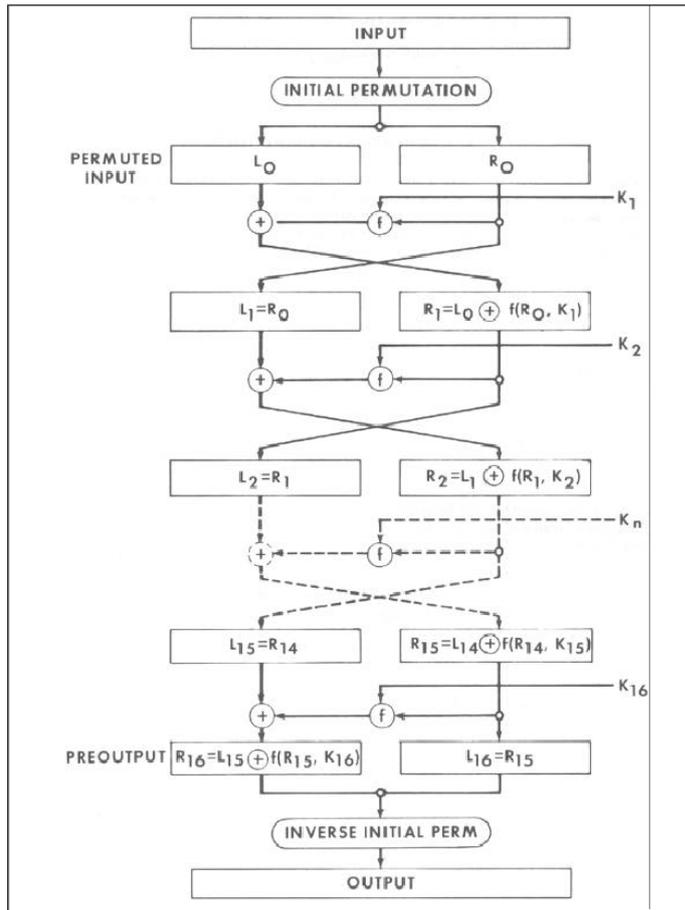


Abbildung 7: Verschlüsselung mit DES, Flowchart [8].

Bit 7 steht. (Siehe Tabelle 2)

Dieser permutierte Block ist dann der Input für die komplexe keyabhängige Berechnung. Der Output aus dieser Berechnung, der Preoutput genannt wird, wird folgender Permutation unterzogen, welche die inverse Permutation der vorherigen war: Der Output beginnt mit Bit 40, gefolgt von Bit 8, dann Bit 48 usw. bis am Ende Bit 25 steht. (Siehe Tabelle 3)

Die Berechnung die den permutierten Input-Block als Input verwendet, um den Preoutput-Block zu erzeugen besteht, mit Ausnahme eines finalen Austausches von Blöcken, aus 16 Iterationen einer Funktion f die auf zwei Blöcken operiert. Einen mit 32 Bits und den anderen mit 48. Dabei produziert sie einen Block von 32 Bits.

Die 64 Bits des Input-Blocks werden in zwei 32 Bits Blöcke L und R unterteilt, was für *Links* und *Rechts* steht. Man teilt also den 64 Bits Block in der Mitte in Links und Rechts auf. LR ergibt somit wieder den gesamten Block.

Tabelle 2: IP ([8], S.10)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabelle 3: IP^{-1} ([8], S.10)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

K sind 48 aus den 64 Bits des Keys. Dann ist der Output $L'R'$ einer Iteration mit Input LR definiert als

$$L' = R \quad (11)$$

$$R' = L \oplus f(R, K) \quad (12)$$

wobei \oplus die XOR-Verknüpfung zwischen den einzelnen Bits ist. (siehe Tabelle 9 auf Seite 64)

Wenn der Output-Block $L'R'$ der 16. Iteration ist, dann ist $R'L'$ der Preoutput-Block. Bei jeder Iteration wird ein anderer Block K aus den 64 Bits des Keys ausgewählt.

Zu den 48 Bits des Schlüssel-Blocks gelangt man durch Anwendung der Funktion KS (Key-Schedule). Diese Funktion erwartet eine natürliche Zahl zwischen 1 und 16 und den 64-Bit Block des Keys und gibt den entsprechenden 48-Bit Block K_n zurück. Das bedeutet somit

$$K_n = KS(n, KEY) \quad (13)$$

Selbstverständlich wird in der n -ten Iteration der Schlüssel-Block K_n generiert.

Nun sei L_0 und R_0 L und R von vorhin, also der permutierte Input-Block, und L_n und R_n sollen L' und R' aus 11 sein, wenn L und R in 11 L_{n-1} und R_{n-1} sind und K K_n ist. Und n ist natürlich zwischen 1 und 16. Vernünftig formuliert und gar nicht so kompliziert wie in Worten beschrieben heißt das somit:

$$L_n = R_{n-1} \quad (14)$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n) \quad (15)$$

Der Preoutput-Block ist dann $R_{16}L_{16}$.

Entschlüsselung Die Entschlüsselung ist die Verschlüsselung in umgekehrter Reihenfolge.

$$R = L' \quad (16)$$

$$L = R' \oplus f(L', K') \quad (17)$$

und

$$R_{n-1} = L_n \quad (18)$$

$$L_{n-1} = R_n \oplus f(L_n, K_n) \quad (19)$$

wobei jetzt $R_{16}L_{16}$ der permutierte Input-Block ist und L_0R_0 der Preoutput-Block. K_{16} ist der Schlüssel der ersten Iteration, K_{15} der der zweiten usw.

Die Funktion f In Abbildung 8 auf Seite 44 sieht man ein Flowchart das die Arbeitsweise der Funktion f darstellt.

E sei dabei eine Funktion, die einen 32 Bits Input-Block nimmt und 48 Bits zurück gibt. Das geschieht nach wie in Tabelle 4:

Wie man sieht, werden einige der Bits doppelt vergeben, wodurch man den Block von 32 auf 48 Bits vergrößert. Die Anordnung der Zeilen und Spalten ist nicht zufällig gewählt. Man ordnet den Block in 8 Blöcke zu je 6 Bits an. Diese 48 Bits werden nun mit den gerade aktuellen 48 Bits des Key-Blocks K (bzw. K_n) XOR-verknüpft. Der

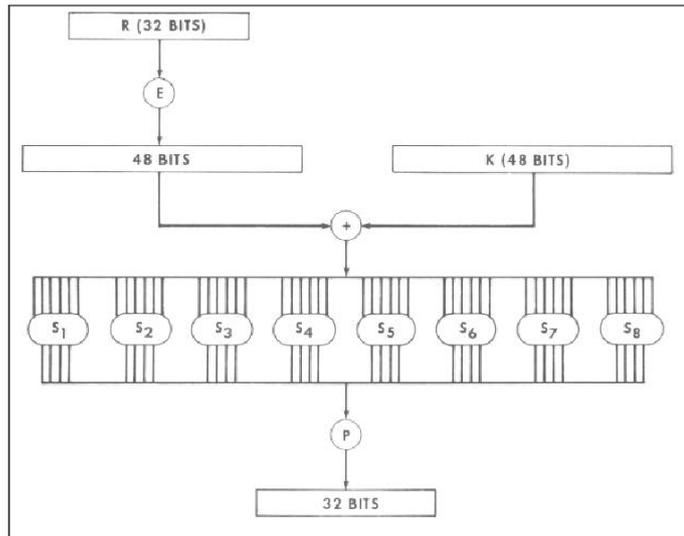


Abbildung 8: Die Funktion f . [8].

Tabelle 4: E ([8], S.13)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Block den man dadurch erhält, wird nun durch 8 Sub-Boxen gejagt. Hier wird nur die erste S-Box behandelt, der Rest verhält sich aber analog. Die erste S-Box, genannt S_1 ist in Tabelle 5.

Ein Block B von 6 Bits wird nun wie folgt mit dieser S-Box ersetzt (bei den anderen analog): Das erste und das Letzte Bit aus B werden als Binärzahl interpretiert die sich klarer Weise zwischen 0 und 3 bewegt. Diese Zahl soll i sein. Die mittleren 4 Bits aus B werden als Binärzahl zwischen 0 und 15 interpretiert. Diese Zahl soll j sein. Nun wird in der S-Box die i -te Reihe und die j -te Spalte nachgeschlagen wo sich eine Zahl zwischen 0 und 15 befindet, die wiederum als vierstellige Binärzahl interpretiert wird. Dieser vierstellige Block ist der Output von S_1 . Ein Beispiel: Wenn der Input 000101 ist, dann wird in Reihe 01, also Reihe 1, und in Spalte 0010, das ist Spalte 2, nachgesehen wo sich die Zahl 7 verbirgt, was dann als Output 0111 ergibt.

Teilt man den Input-Block in 8 Blöcke zu je 6 Bits ein, ist also B_1, B_2, \dots, B_8

Tabelle 5: S_1 ([8], S.14)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

der gesamte Block, so wird die S-Box S_i dabei auf den Block B_i angewendet. Das Endergebnis der Funktion f ist schließlich die Anwendung der Funktion P auf diese substituierten Blöcke B_1 bis B_8 , oder anders formuliert $P(S_1(B_1)S_2(B_2)\dots S_8(B_8))$. Die Funktion P ist wieder eine Permutation die wie in Tabelle 6 funktioniert:

Tabelle 6: P ([8], S.15)

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Das sechzehnte Bit des Input-Blocks in P wird an die erste Stelle gesetzt. An die zweite das siebte, dann das zwanzigste usw. Was hier herauskommt ist das Ergebnis der Funktion f . [8]

3.2 Advanced Encryption Standard, Rijndael

Rijndael (gesprochen „Reindahl“) war der überraschende Gewinner des Wettbewerbs, der den neuen Advanced Encryption Standard (AES) hervorbringen sollte. Der Wettbewerb wurde vom National Institute for Standards and Technology (NIST) der Vereinigten Staaten von Amerika zwischen 1997 und 2000 abgehalten. *Überraschend* gewonnen hat Rijndael, weil zu diesem Zeitpunkt niemand dachte, dass ein ausländisches Team gewinnen wird. Seine Entwickler, Vincent Rijmen und Joan Daemen, sind nämlich beide Belgier.

Der Prozess zur Findung des neuen AES, der DES und triple-DES ablösen sollte, war komplett offen. Das NIST hat lediglich die Plattform bereitgestellt und anschließend jeden eingeladen Vorschläge zu bringen und Vorschläge anderer Teilnehmer zu

evaluieren. Das hat natürlich Vorteile. Jeder der einen kryptographischen Algorithmus einschickt, versteht etwas von Kryptographie. Weiters will jeder gewinnen und wird somit sein oder ihr Bestes geben, Fehler und Designschwächen in anderen Algorithmen aufzudecken.

Rijndael gehört zu den vorhin schon kurz erwähnten Block-Cipher-Algorithmen. Das heißt, es wird ein Block bestimmter Länge von Klartext zu einem Block gewisser Länge im Ciphertext übersetzt. Bei AES haben diese Blöcke eine Größe von 128 Bits. Ist der Klartext größer als ein Block, muss er in mehrere Blöcke aufgeteilt werden. Wenn ein Block übrig bleibt, der kleiner ist als die von AES verarbeitete Blockgröße, wird er auf die benötigte Größe mit Bits befüllt.

Oft wird AES synonym mit Rijndael verwendet. Das ist aber nicht ganz korrekt. Rijndael ist in der Lage, variable Block- und Key-Längen zu verarbeiten, solange sie zwischen 128 und 256 Bits und ein Vielfaches von 32 sind. In den FIPS-Standard für AES wurden nur die Key-Längen 128, 192 und 256 aufgenommen und die Datenblockgröße ist auf 128 beschränkt. [9] Alle übrigen möglichen Längen wurden dafür nicht evaluiert. Außerdem ist Rijndael theoretisch in der Lage, auch größere Block- und Key-Längen zu verarbeiten. Zur Zeit gibt es dafür aber keinen Bedarf. [2] In weiterer Folge wird hier die Arbeitsweise von AES beschrieben. Hierfür soll ein Block Daten Schritt für Schritt verschlüsselt werden.

3.2.1 Funktionsweise von AES

Zunächst brauchen wir einen Block Daten und einen Schlüssel. Der Datenblock soll in_i heißen und 128 Bits beinhalten. Der Schlüssel soll ck heißen und ebenfalls 128 Bits lang sein. Der Index i im Datenblock indiziert jedes Byte (nicht Bit) und geht somit von 0 bis 15. Der AES-Algorithmus operiert intern immer auf *State*-Arrays. Das sind zweidimensionale Arrays mit vier Zeilen und je nach Blocklänge vielen Spalten. Die Spaltenanzahl ist Nb , wobei Nb gleich die Blocklänge dividiert durch 32 ist. Bei 128 ist Nb somit ebenfalls vier und das State-Array eine quadratische Matrix der Dimension 4×4 . Das State-Array soll das Symbol $s_{r,c}$ erhalten, wobei r der Zeilen- und c der Spaltenindex ist und sich jeweils zwischen 0 und 3 bewegen. Bei größeren Blockgrößen wird die Obergrenze des Spaltenindex natürlich größer. Zu erst muss somit das Input-Array in das State-Array kopiert werden. Der Cipher oder inverse Cipher wird dann auf dieses State-Array angewendet und das Ergebnis in das Output-Array out_i kopiert. Das Output-Array ist wie das Input-Array eindimensional und indiziert die Bytes

durch von 0 bis 15. Das Hin- und Herkopieren erfolgt nach folgender Vorschrift:

$$s_{r,c} = in_{r+4c}, \text{ für } 0 \leq r < 4 \text{ und } 0 \leq c < N_b. \quad (20)$$

$$out_{r+4c} = s_{r,c}, \text{ für } 0 \leq r < 4 \text{ und } 0 \leq c < N_b. \quad (21)$$

Grafisch wird dies in Abbildung 9 veranschaulicht. [9]

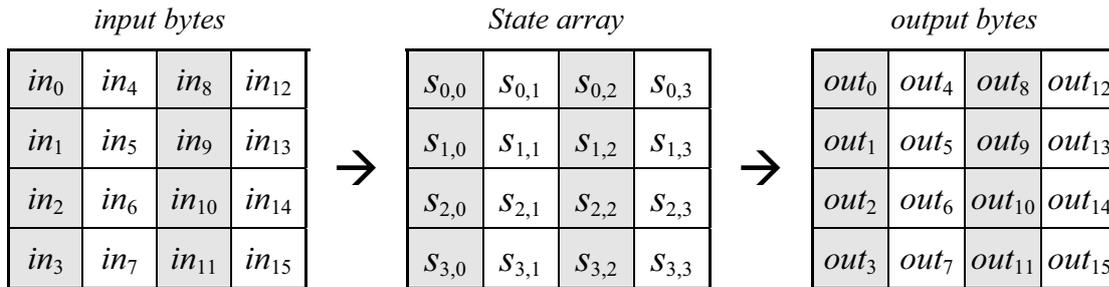


Abbildung 9: In- und Output des State-Arrays [9]

Auch der Schlüssel wird in ein zweidimensionales Array mit vier Zeilen übertragen. Die Anzahl der Spalten wird mit N_k bezeichnet und ist, wie beim State-Array, gleich der Schlüssellänge dividiert durch 32. Wenn der Schlüssel $z_0, z_1, z_2, z_3 \dots z_{4 \cdot N_k - 1}$ entspricht, dann wird er wie folgt in das Key-Array übertragen:

$$k_{i,j} = z_{i+4j}, 0 \leq i < 4, 0 \leq j < N_k \quad (22)$$

wobei i und j Zeilen bzw. Spaltenindex des Schlüsselarrays sind. Dieses Schlüsselarray ist im Listing 11 `CipherKey`.

Jetzt wo wir wissen, wie ein State aufgebaut ist und wie wir eines aus einem anfänglich unverschlüsselten String von Bits erzeugen, können wir den Algorithmus Schritt für Schritt durchgehen und uns ansehen, was passiert. Wenn wir das Listing 11 ansehen, sehen wir eine Funktion namens „Aes“, der das Anfangs-State und der Cipher Key übergeben wird. Diese Funktion ist, wenn man so will, die Main-Funktion des Ganzen. Von hier aus werden dann weitere Funktionen aufgerufen.

Die erste Funktion, die in Zeile 3 aufgerufen wird, ist `KeyExpansion` die den Schlüssel und einen Pointer auf das Array `ExpandedKey` übernimmt. Der Algorithmus, der beschreibt, wie man die Runden- (Round) Keys aus dem ursprünglichen Schlüssel erzeugt heißt „Key Schedule“. Die Aneinanderkettung aller Rundenschlüssel heißt „Expanded Key“.

Listing 11: Algorithmus zum Verschlüsseln mit AES [2]

```

1 Aes(State, CipherKey)
2 {
3     KeyExpansion(CipherKey, ExpandedKey);
4     AddRoundKey(State, ExpandedKey[0]);
5     for(i=1; i < Nr; i++)
6         Round(State, ExpandedKey[i]);
7     FinalRound(State, ExpandedKey[Nr]);
8 }

```

In Zeile 4 wird zum ersten mal `AddRoundKey` aufgerufen. Dieser Funktion wird das State-Array und der erste Rundenschlüssel übergeben. Die Anzahl der Runden wird mit N_r bezeichnet und ist von der Block- und Schlüssellänge abhängig. Nach dem ersten Aufruf von `AddRoundKey` wird die Funktion `Round` $N_r - 1$ -mal aufgerufen. Was in `Round` passiert, sieht man in Listing 12. Nachdem die Runden abgearbeitet wurden, wird zum Schluss `FinalRound` ausgeführt. [2]

Listing 12: Rundentransformationen [2]

```

1 Round(State, ExpandedKey[i])
2 {
3     SubBytes(State);
4     ShiftRows(State);
5     MixColumns(State);
6     AddRoundKey(State, ExpandedKey[i]);
7 }
8
9 FinalRound(State, ExpandedKey[Nr])
10 {
11     SubBytes(State);
12     ShiftRows(State);
13     AddRoundKey(State, ExpandedKey[Nr]);
14 }

```

Die erste Funktion, die hier näher betrachtet werden soll, ist `AddRoundKey`. Alles was hier passiert ist, dass jedes Bit des aktuellen States mit dem aktuellen Rundenschlüssel XOR-addiert⁸ wird. Sei also $(a_{i,j})$ die Matrix des aktuellen States und $(k_{i,j})$ die des Rundenschlüssels, so ist $(b_{i,j})$ die Ergebnismatrix aus $(a_{i,j}) \oplus (k_{i,j}) = (b_{i,j})$. Dabei ist das Element l, m der Matrix $(b_{i,j})$ gleich $a_{l,m} \oplus k_{l,m}$.

⁸Siehe Tabelle 9 auf Seite 64

Die Anzahl der Runden ist von der Block- und Schlüsselgröße abhängig. Verschiedene kryptoanalytische Überlegungen haben dazu geführt, dass bei Rijndael die Rundenanzahl wie in Tabelle 7 bestimmt wird.

Tabelle 7: Die Anzahl an Runden (N_r) als Funktion von N_b ($N_b = \text{Blocklänge}/32$) und N_k (Schlüssellänge / 32)[2].

N_k	N_b				
	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

Wie zuvor erwähnt, heißt die Routine, die den Expanded Key aus dem Cipher Key generiert *Key Schedule*. In Key Schedule wird zudem auch der Rundenschlüssel selektiert. Die Gesamtanzahl an Bits in ExpandedKey ist gleich der Blocklänge mal der Rundenanzahl plus 1. Falls sich jemand beim Listing weiter oben gewundert hat, dass man um einen Rundenschlüssel zu wenig hat, da der erste in Zeile 4 schon verbraucht wird. Deshalb das plus 1.

Die Funktion `Aes` wäre damit abgehakt. In der Unterfunktion `Round` sind aber noch ein paar Funktionen offen. Die erste, die wir sehen, ist in Zeile 3 des Listings `12 SubBytes`. `SubBytes` ist im Prinzip ganz einfach. Jedes Byte im State wird durch ein entsprechendes Byte in der S-Box ersetzt. Die S-Box erhält man wie folgt:

Aufgeteilt in zwei Transformationen

1: Das multiplikative Inverse im endlichen Feld $GF(2^8)$ des in Polynomialdarstellung übersetzten Bytes $a = a_7a_6...a_1a_0 \neq 0$ wird bestimmt. Die Koeffizienten des Inversen sind das Ergebnisbyte $b = b_7b_6...b_1b_0$. Das Element 00 wird auf sich selbst abgebildet.

2: Die folgende affine Transformation (über $GF(2)$) wird angewendet:

$$c_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (23)$$

für $0 \leq i < 8$, wobei b_i das i -te Bit des Bytes ist, c_i das i -te Bit des Bytes c mit dem Wert `0x63`. Der Operator \oplus ist der XOR-Operator. Die S-Box in hexadezimaler Form ganz ausgeschrieben ist in Tabelle 8 zu sehen.

Das zu ersetzende Byte `xy` wird in zwei Hälften geteilt, `x` und `y`. Dann wird in

Tabelle 8: S-Box in der Routine `SubBytes`. [9]

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

der Tabelle die Zeile x und die Spalte y heraus gesucht und das Byte im State mit dem entsprechenden Byte der S-Box ausgetauscht. Wenn zum Beispiel das Byte des State-Arrays das gerade ersetzt werden soll $s_{0,2} = 3d$, dann schaut man in der Tabelle in die Zeile 3 und in die Spalte d, und man erhält 27. Somit ist $s'_{0,2} = 27$.

Der nächste Schritt ist `ShiftRows`. Auch diese Operation ist recht unkompliziert. Alles was gemacht wird, ist die Spalten des State zyklisch nach links zu verschieben. Die Anzahl der Schritte, um die verschoben wird hängt dabei von der Zeile ab. Hier sagt ein Bild mehr als tausend Worte. Abbildung 10⁹ auf Seite 51 zeigt, wie die Zellen des State-Arrays verschoben werden.

Der vorletzte Schritt ist `MixColumns`. Was passiert ist folgendes: Sei b das State-Array nach dem `MixColumns`-Schritt, a das State-Array davor und c eine Matrix mit konstanten Werten. b erhält man dann wie in Gleichung 24 angegeben.

$$\begin{bmatrix} b_{0,p} \\ b_{1,p} \\ b_{2,p} \\ b_{3,p} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,p} \\ a_{1,p} \\ a_{2,p} \\ a_{3,p} \end{bmatrix} \text{ für } 0 \leq p < N_b. \quad (24)$$

⁹Quelle: <http://de.wikipedia.org/wiki/Datei:AES-ShiftRows.svg>

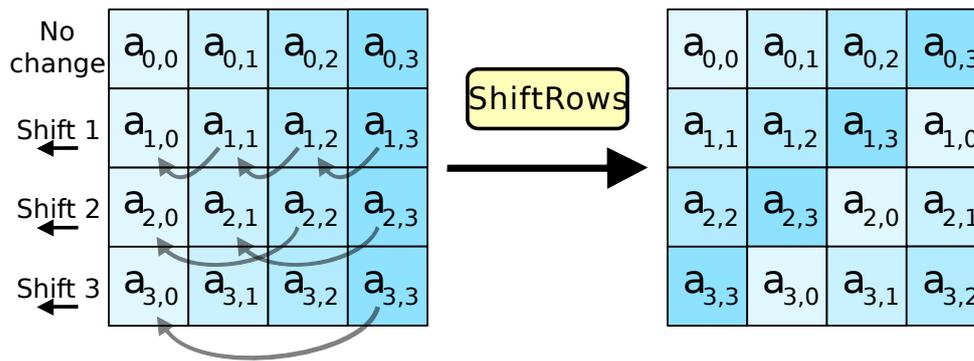


Abbildung 10: Die zellenweise Vertauschung in ShiftRows.

Oder anders ausgedrückt:

$$b_{0,p} = (02 \bullet a_{0,p}) \oplus (03 \bullet a_{1,p}) \oplus a_{2,p} \oplus a_{3,p} \quad (25)$$

$$b_{1,p} = a_{0,p} \oplus (02 \bullet a_{1,p}) \oplus (03 \bullet a_{2,p}) \oplus a_{3,p} \quad (26)$$

$$b_{2,p} = a_{0,p} \oplus a_{1,p} \oplus (02 \bullet a_{2,p}) \oplus (03 \bullet a_{3,p}) \quad (27)$$

$$b_{3,p} = (03 \bullet a_{0,p}) \oplus a_{1,p} \oplus a_{2,p} \oplus (02 \bullet a_{3,p}) \quad (28)$$

Das waren die Schritte zum Verschlüsseln mit AES. Die letzte Runde, die mit FinalRound abgearbeitet wird unterscheidet sich von den normalen Runden nur dadurch, dass MixColumns nicht ausgeführt wird. [9]

3.2.2 Unterrichtssequenz

Ein nettes kleines Programm, das AES intern verwendet, ist aespipeline¹⁰. Wie der Name vermuten lässt, kann man recht einfach Klartextströme nach aespipeline pipen, was den Ciphertext auf STDOUT ausgibt bzw. umgekehrt Ciphertext nach aespipeline pipen, was den Klartext ausgibt. Gleichzeitig ist dies auch eine gute Übung um mit den üblichen Unix-Tools umzugehen und Datenströme umleiten zu lernen. Die Schülerinnen und Schüler sollen wie schon bei der RSA-Sequenz in Zweier-Teams zusammengehen. Zunächst müssen sie sich auf einen gemeinsamen Schlüssel einigen. Wie sie das machen, bleibt ihnen überlassen. Sie können sich eine Passphrase ausdenken, eine Datei wählen, in der das Passwort steht, sich gegenseitig ein Passwort mit GPG verschlüsselt schicken, oder was ihnen selbst einfällt. Die Passphrase muss bei aespipeline mindestens 20 Zeichen lang sein. Um noch ein weiteres Unix-Werkzeug zu verwenden, wollen wir

¹⁰<http://loop-aes.sourceforge.net/>

zunächst ein Passwort aus einer Datei erzeugen. Diese Datei ist in meinem Fall ein PDF namens „vorwort.pdf“. Um ein 20 Zeichen langes Passwort aus vorwort.pdf zu erhalten, kann man zum Beispiel wie folgt vorgehen:

Listing 13: Passwort aus einer Datei erzeugen

```
stefan:~$ base64 vorwort.pdf | head -c 20 > pw.txt
```

Was `base64` macht, wissen wir schon. Es encodiert die Datei in ASCII-Zeichen. Diesen viel länger als 20 Zeichen langen Zeichenstrom leiten wir weiter an `head`. Dieses Programm gibt normaler Weise die ersten zehn Zeilen einer Datei aus. Mit dem Schalter `-n` kann man diese Zahl variieren. Mit `-c` kann man die Anzahl der Bytes bestimmen, die ausgegeben werden sollen. Da ein Buchstabe im ASCII-Zeichensatz durch sieben Bits codiert ist und das achte stets 0 ist, erhält man mit `-c 20` exakt 20 Zeichen. Diese 20 Zeichen werden dann in die Datei `pw.txt` umgeleitet. In der Realität könnten sich zwei Kommunikationspartner ausmachen, ihre Passwörter täglich zu rotieren und sie auf folgende Art zu generieren: Jeden Tag zu einer bestimmten Uhrzeit besuchen beide Teilnehmer eine Website die stark frequentiert wird und sich täglich ändert. Zum Beispiel `http://news.google.com/`. Das dritte Bild auf der Seite soll abgespeichert werden und die nächsten 35 Bytes angefangen vom Byte Nummer „Tag des Jahres“ (also z.B. 15. Jänner ist 15) sind der Schlüssel. Solcherlei Tricks sind zwar recht praktisch, aber stellen ein Problem dar: Man kann sich nicht sicher sein, wie gut die Zufälligkeit in der durch sie erzeugten Bitsequenz ist. Eine bessere, wesentlich zufälliger Möglichkeit an 20 Bytes zu gelangen, ist natürlich mittels

Listing 14: Passwort aus einer Datei erzeugen (zufälliger)

```
stefan:~$ dd if=/dev/random of=./pw.txt count=20 bs=1
```

Nun kann eine beliebige Datei verschlüsselt werden. Dazu macht man folgendes:

Listing 15: Datei mit aespipeline verschlüsseln

```
stefan:~$ aespipeline -e aes128 -P pw.txt < orly.txt > ciphertxt
```

Hier wird die Datei `orly.txt` mit dem Passwort in `pw.txt` verschlüsselt. Zum Einsatz kommt AES mit 128 Bit-Key und der Ciphertext wird in die Datei `ciphertxt` umgeleitet. Versenden kann man dann zum Beispiel wieder mittels `base64` und E-Mail, oder hängt die verschlüsselten Bits an ein Bild an und verschickt das Bild, um zu verschleiern, dass überhaupt etwas sinnvolles geschickt wird.

Eine weitere Anwendung von AES findet sich bei verschlüsselten Partitionen von Festplatten. Linux bietet hier für das Tool `cryptsetup` in Verbindung mit den Crypto-Kernel-Modulen an. Mit `cryptsetup` können auf Partitionen, oder allem was sich gleichwertig behandeln lässt, zum Beispiel logische Volumes oder Loop-Devices, Cryptocontainer erstellt werden. Diese können dann mit `cryptsetup` geöffnet werden und erstellen daraufhin einen Device-Node, der sich wie ein normales blockorientiertes Gerät verhält, eben wie eine Partition. Jedoch werden alle Daten die darauf geschrieben werden automatisch verschlüsselt und alle die gelesen werden, entschlüsselt. Für die folgenden Aktivitäten benötigt man Root-Rechte, es bietet sich also an, das Ganze in einer virtuellen Maschine ablaufen zu lassen.

Als erstes brauchen wir etwas, das unseren Container beinhalten soll. Da ich im Augenblick keine leere Partition zur Hand habe, erstelle ich einfach eine 100 MB große Datei und binde diese als Loop-Device ein:

Listing 16: Datei erzeugen

```
stefan:~$ dd if=/dev/zero of=./cryptoarchiv bs=1024 \  
count=$(( 1024 * 100 ))
```

Als nächstes muss diese Datei mit einem Loop-Device-Node verbunden werden. Wie man sieht, braucht man dafür Root-Rechte.

Listing 17: Datei mit Loop-Device verbinden

```
root@zero:~# losetup /dev/loop0 ./cryptoarchiv
```

Auf diesem Device kann man jetzt, wenn man möchte, eine Partition erstellen. Notwendig ist dies allerdings nicht. Wir wollen hier im nächsten Schritt gleich einen Cryptoluks-Container erzeugen:

Listing 18: Cryptoluks-Container erzeugen

```
root@zero:~# cryptsetup luksFormat -c aes-cbc-essiv:sha256  
--key-size 128 /dev/loop0  
  
WARNING!  
=====  
This will overwrite data on /dev/loop0 irrevocably.  
  
Are you sure? (Type uppercase yes): YES  
Enter LUKS passphrase:
```

```
Verify passphrase:
Command successful.
```

Die Optionen bedeuten, dass als Algorithmus AES mit Cipher Block Chaining verwendet werden soll. Das „essiv“ wird hinzugefügt, um Watermark-Attacken zu erschweren. Die Schlüssellänge soll 128 Bits betragen und das Device ist unser vorheriges Loop-Device. Zur Bestätigung muss man „YES“ in Großbuchstaben eintippen, gefolgt von der gewünschten Passphrase. Es wird keine Rückmeldung über die Eingabe der Passphrase gegeben und man muss sie zwei Mal eingeben. Alternativ kann man auch eine Schlüsseldatei verwenden die dann als letztes Argument übergeben wird. In jedem Fall gilt aber, Passphrase wirklich gut merken, oder auf die Schlüsseldatei wirklich gut aufpassen.

Der nächste Schritt ist, den Crypto-Container ins System zu integrieren. Dazu macht man folgendes:

Listing 19: Cryptoluks-Container erzeugen

```
root@zero:~# cryptsetup luksOpen /dev/loop0 cryptocontainer
Enter LUKS passphrase:
key slot 0 unlocked.
Command successful.
```

Das Argument „cryptocontainer“ ist beliebig. Es gibt nur an, wie die Datei unter `/dev/mapper/` heißen soll, die jetzt als blockorientiertes Gerät angesprochen werden kann, und beim Schreiben bzw. Lesen AES verwendet. Da unser Container noch kein Dateisystem hat, erstellen wir eines:

Listing 20: Cryptoluks-Container formatieren

```
root@zero:~# mkfs.ext2 /dev/mapper/cryptocontainer
```

`/dev/mapper/cryptocontainer` kann jetzt ganz normal gemountet werden:

Listing 21: Cryptoluks-Container formatieren

```
root@zero:~# mkdir /mnt/cryptocontainer
root@zero:~# mount /dev/mapper/cryptocontainer \
    /mnt/cryptocontainer/
```

Ab jetzt ist alles so wie gehabt. Alles was in das Verzeichnis `/mnt/cryptocontainer` kommt, wird über das Mapper-File `/dev/mapper/cryptocontainer` verschlüsselt und

an das Loop-Device `/dev/loop0` weitergereicht, was wiederum mit der Datei `cryptoarchiv` verbunden ist. Will man die Verbindung wieder lösen, geht man genau umgekehrt vor:

Listing 22: Cryptoarchiv unmounten

```
root@zero:~# umount /mnt/cryptocontainer/  
root@zero:~# cryptsetup luksClose cryptocontainer  
root@zero:~# losetup -d /dev/loop0
```

Das war es auch schon. Diese Aufgaben lassen sich natürlich auch wunderbar in Bash-Skripte schreiben, sodass ein komfortabler Umgang mit verschiedenen Crypto-Containern erreicht werden kann.

3.3 Twofish

Twofish wurde wie Rijndael ins Rennen um den Platz des neuen AES geschickt. Entwickelt und getestet von seinen Schöpfern Bruce Schneier, Niels Ferguson, John Kelsey, Doug Whiting, David Wagner und Chris Hall erfüllte er alle Kriterien zur Teilnahme und schaffte es sogar in die Runde der besten fünf. Diese Kriterien waren 128-Bit symmetrischer Block Cipher, Schlüssellängen von 128, 192 und 256 Bits, keine schwachen Schlüssel, hohe Effizienz auf der Intel Pentium Pro-Plattform und anderen Software- und Hardware-Plattformen, flexibles Design (zusätzliche Key-Längen, auf vielen Plattformen implementierbar, verwendbar für Stream Cipher, Hash-Funktionen und MAC) und ein simples Design, damit er einfach analysiert werden konnte. Zusätzlich haben sich die Entwickler noch weitere Kriterien auferlegt, unter anderem beliebige Schlüssellängen bis 256 Bits zu akzeptieren und Ineffizienz auf diversen 8 bis 64-Bit Mikroprozessoren zu vermeiden. Außerdem sollte Twofish zur Verwendung in Einweg-Hash-Funktionen und Pseudo-Zufallszahlengeneratoren geeignet sein. [14]

3.3.1 Funktionsweise von Twofish

Ähnlich wie Rijndael und viele andere Algorithmen, ist auch Twofish aus verschiedenen Bauteilen zusammengesetzt, die jeweils eine oder mehrere bekannte Attacken auf ihn kontern sollen und, wenn möglich, auch zukünftige Angriffe abwehren können. Der erste Baustein ist ein sog. Feistel Network, welches schon im Abschnitt DES (Seite 38) vorkam, dort aber noch nicht so benannt wurde.

Feistel Networks Feistel Networks, im deutschen oft Feistelchiffre genannt, sind nach ihrem Erfinder Horst Feistel benannt, der sie für den Entwurf von Lucifer ent-

wickelte, ein von IBM entwickelter Verschlüsselungsalgorithmus.

Ein Feistel Network ist im allgemeinen eine Methode um eine beliebige Funktion in eine Permutation umzuwandeln. Diese Funktion nennt man meistens die F-Funktion. Die F-Funktion bildet einen Eingabe-String auf einen Ausgabe-String, abhängig von einem Schlüssel, ab. Diese F-Funktion ist immer nicht-linear und kann unter Umständen nicht alle Ausgaben, die prinzipiell im Ausgaberaum möglich sind, ausgeben. Eine allgemeine Form so einer F-Funktion kann man wie folgt notieren:

$$F : 0, 1^{n/2} \times 0, 1^N \rightarrow 0, 1^{n/2} \quad (29)$$

Dabei ist n die Blockgröße des Feistel Networks. F ist die Funktion, die $n/2$ Bits dieses Blocks nimmt, N Bits eines Keys, dies miteinander kombiniert und $n/2$ Bits Output produziert. Genauer ist es so, dass der Block, ähnlich wie im auf Seite 41 beschriebenen Abschnitt von DES, in eine linke und rechte Hälfte geteilt wird und dann eine Hälfte aus einer Runde mit dem Schlüssel kombiniert mit der anderen Hälfte der vorherigen Runde XOR-verknüpft wird und im Anschluss die Hälften um eins nachrücken, das heißt die Hälfte die jetzt die aktuelle war, ist die aus der vorherigen und die nächst ist die jetzige usw. bis alle Bits, aller Input-Blöcke einmal geändert wurden. Dabei sind zwei Runden ein Zyklus und in einem Zyklus wurde jedes Bit eines Blocks genau einmal geändert.

S-Boxen Der nächste Bauteil sind S-Boxen, die auch schon im Abschnitt AES auf Seite 49 vorkam. Twofish verwendet vier verschiedene, bijektive 8-mal-8 Bits S-Boxen, die mit zwei fixen 8-mal-8 Bits Permutationen und Teilen des Schlüssels gebaut werden.

MDS Matrizen Maximum distance separable (MDS) Matrizen sind Matrizen, die eine Abbildung von a Feldelementen auf b Feldelemente beschreibt, die einen zusammengesetzten Vektor mit $a + b$ Elementen produziert. Dieser Vektor hat die Eigenschaft, dass die Anzahl der Elemente, die unterschiedlich sind, zwischen zwei beliebigen unterschiedlichen Vektoren, die mit einer MDS-Abbildung produziert wurden, mindestens $b + 1$ ist. Diese MDS-Matrix besteht aus $a \times b$ Elementen. Benötigt werden diese Eigenschaften um Diffusion des Klartexts im Ciphertext zu erzeugen. Das heißt, dass bei kleinen Änderungen im Klartext, große Änderungen, am besten zufällig aussehende, im Ciphertext passieren.

Pseudo-Hadamard Transformationen Eine n -Bit pseudo-Hadamard Transformation (PHT) erzeugt für die Eingaben a und b die Ausgaben $a' = a + b \bmod 2^n$ und $b' = a + 2b \bmod 2^n$. In Twofish werden 32-Bit PHT verwendet, um die Ausgaben seiner zwei parallelen 32-Bit g -Funktionen zu vermischen. Was die g -Funktion ist, kommt später.

Whitening Hierbei wird ein Schlüssel vor der ersten und nach der letzten Runde mit einem zu verschlüsselndem Block XOR-verknüpft. Der Schlüssel für die pre-Runden-Verknüpfung und der für die post-Runden-Verknüpfung werden auf die gleiche Art wie die Runden-Schlüssel erzeugt.

Key Schedule Wie schon bei Rijndael ist auch bei Twofish die Key Schedule die Methode, mit der aus dem Schlüssel, Runden-Schlüssel erzeugt werden. In Twofish ist diese recht kompliziert, da viele Runden-Schlüssel benötigt werden.

Jetzt kennen wir fast alle Schritte, die in Twofish der Reihe nach angewendet werden. Was jetzt noch fehlt sind Rotationen um n Bits. Die sind aber recht einfach: Ein Wort, das aus r Bits besteht, wobei $n \geq r$, wird um n Bits nach links so rotiert, indem man jedes Bit um n Stellen nach links verschiebt, wobei Bits die auf der linken Seite des Worts „hinausfallen“, auf der rechten Seite wieder hinein kommen. Bei Rotation nach rechts ist es natürlich genau umgekehrt. Alle Schritte werden jetzt nochmal erklärt. Dabei ist Flussdiagramm in Abb. 11 sehr hilfreich.

Der Plaintext (P) wird zunächst in Blöcke zu je 128 Bits unterteilt. Nun folgt die Verschlüsselung eines solchen Blocks.

1: Aufteilung in vier Wörter Der Plaintext wird in vier 32-Bits Wörter aufgeteilt.

2: Input Whitening Jedes dieser Wörter wird mit je einem Key-Wort XOR-verknüpft.

3: 16 Runden In jeder Runde passiert folgendes:

- i: g-Funktions-Input** Die zwei Wörter auf der linken Seite werden als Input für die g -Funktion verwendet. Zuvor wird das zweitlinkeste um 8 Bits nach links rotiert.
- ii: g-Funktion** Beide linken Wörter werden nun durch je vier, ein Byte lange S-Boxen geschickt und anschließend mit einer MDS-Matrix vermischt.
- iii: g-Funktions-Output** Die beiden Outputs aus der g -Funktion werden mit einer PHT miteinander kombiniert und zwei Schlüsselwörter werden hinzugefügt.

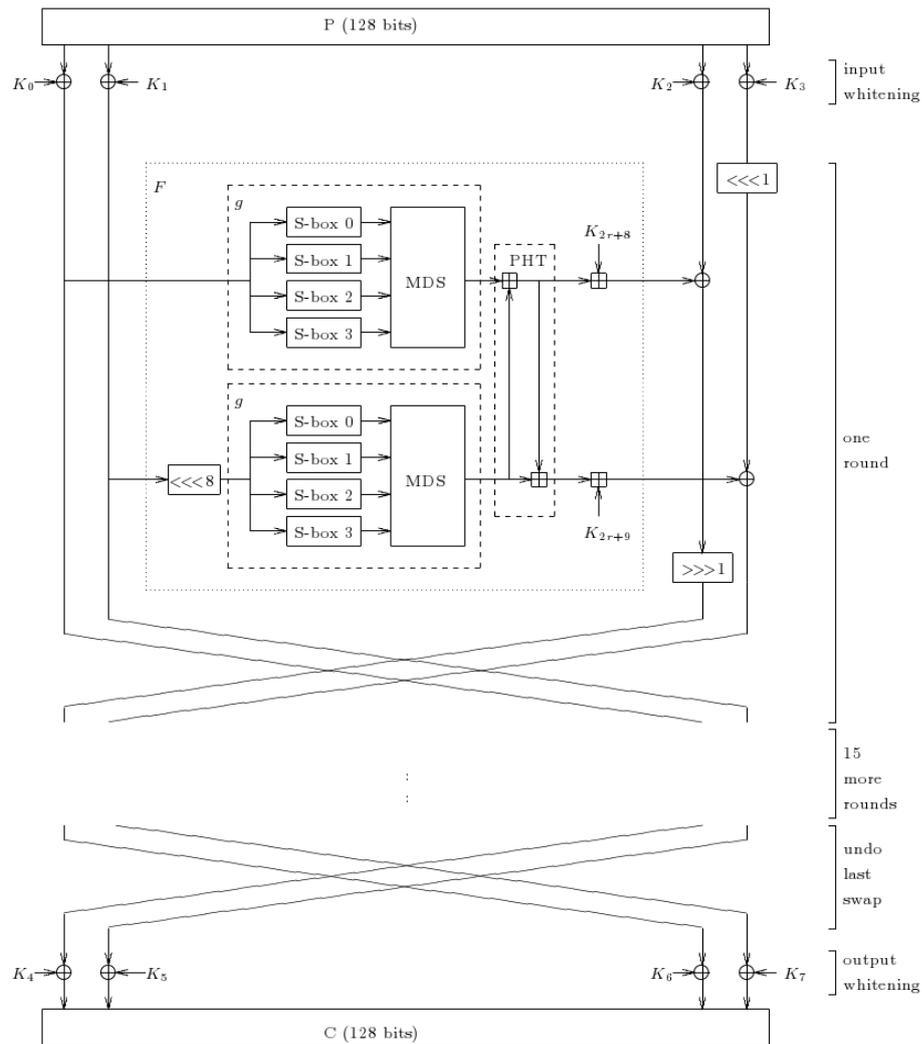


Abbildung 11: Verschlüsselung mit Twofish, Flowchart [14].

iv: XOR mit Rechts Anschließend wird je ein linkes mit einem rechten Wort XOR-verknüpft, wobei eines vorher noch um ein Bit nach links und das andere nachher um ein Bit nach rechts rotiert wird.

v: Nächste Runde Für die nächste Runde werden die rechte und die linke Seite vertauscht.

4: Letzte Vertauschung Nach 16 Runden wird links und rechts noch einmal vertauscht.

5: Output Whitening Jedes der vier Teilwörter wird noch einmal mit je einem Key-Wort XOR-Verknüpft.

So wurden die 128 Bits des Plaintexts in 182 Bits des Ciphertexts umgewandelt. [14]

Wie genau die einzelnen Komponenten wiederum aufgebaut sind und wie das Key-Scheduling funktioniert, kann in [14] genau nachgelesen werden.

3.3.2 Unterrichtssequenz

Beim Ausarbeiten des Twofish-Abschnitts hatte ich das Gefühl, dass Twofish um einiges komplizierter ist als Rijndael, was auch ein Kritikpunkt an Twofish und ein Grund für die Wahl von Rijndael zum neuen AES war. Dennoch kann man mit Twofish sehr gut zeigen, wie Verschlüsselungsalgorithmen aufgebaut sind. Dabei kann man ruhig auf die genaue Funktionsweise der einzelnen Bauteile nur oberflächlich eingehen, eventuell kurz erklären, um was es geht (Substitution nach einem bestimmten Schema etc.) und ansonsten als Blackbox lassen. Damit kann man den stark modularen Charakter von Twofish ausnutzen, um Algorithmen-Design zu vermitteln, aber umgeht die hohe Komplexität des Ciphers.

Eines der zahlreichen Softwareprodukte, die Twofish verwenden, ist TrueCrypt¹¹. Leider ist aufgrund der historischen Entwicklung von TrueCrypt ein Lizenz-Wirr-Warr entstanden, dass durch die TrueCrypt Collective License unter einen Hut gebracht wurde, aber weder von der Free Software Foundation als Freie Software Lizenz, noch von der OSI¹² als Open Source deklariert wurde. Aus diesem Grund verzichteten auch große GNU/Linux-Distributionen darauf, TrueCrypt in ihren Repositories anzubieten und raten von der Verwendung ab. Zumindest ist der Quellcode aber von jedem einsehbar, (und teilweise auch durchaus freie Software) was schonmal ein enormer Vorteil gegenüber Closed-Source Software ist. Man muss sich also überlegen, ob einem die technischen Vorteile, die TrueCrypt bietet, dieser Verzicht auf Freiheit wert sind.

Einer der technischen Vorteile ist, dass TrueCrypt sowohl für MS Windows, Apple Mac OS X und GNU/Linux verfügbar ist. Wenn man die Anforderung hat, dass man zwischen mindestens zwei dieser Plattformen interoperabel bleiben muss, hat man außer TrueCrypt nicht viel Auswahl.

TrueCrypt kann Container-Files erstellen, oder ganze Partitionen und Speichermedien verschlüsseln. Wenn man ein nicht vollverschlüsseltes System verwendet und mit verschlüsselten Containern, Partitionen, USB-Sticks usw. arbeitet, sollte man sich im Klaren darüber sein, dass die an sich verschlüsselten Daten eventuell in eine Ausla-

¹¹<http://www.truecrypt.org/>

¹²Open Source Initiative

gerungsdatei, Swap-Partition oder temporäre Dateien ausgelagert werden und damit nicht mehr geschützt sind.

Installation (GNU/Linux) Da, wie gesagt, TrueCrypt nicht in den Repositories der Distributionen zu finden ist, muss man es von Hand installieren. Dafür lädt man sich ein Tar.gz-Archiv für die eigene Architektur (32- oder 64-Bit Intel) herunter und entpackt dieses. Dadurch erhält man eine ausführbare Datei namens `truecrypt-6.3a-setup-x64`. Wegen akuter Paranoia starte ich dieses aber nur als normaler Benutzer und wähle im erscheinenden Fenster den Punkt **Extract .tar Package File**. Damit erhält man nach Zustimmung zur TrueCrypt-Lizenz ein Tar-Archiv, das das TrueCrypt-Binary enthält. Dieses kopierte ich anschließend händisch an einen vernünftigen Ort, nämlich `/usr/bin`. Nun startete ich als normaler User das Programm indem ich in einem Terminalemulator `truecrypt` eingebe. Alternativ kann man sich auch Buttons für KDE, Gnome und Konsorten erstellen. Damit startet das Programm und man sieht das Anfangsfenster. (Siehe Abb. 12)

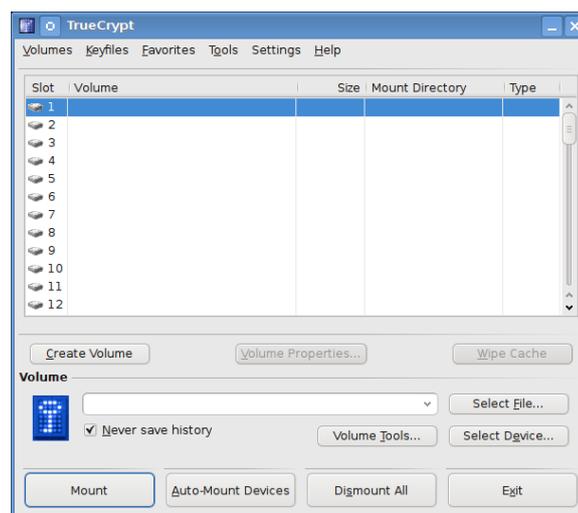


Abbildung 12: Anfangsfenster von TrueCrypt.

In diesem Fenster sieht man, dass man 64 Slots hat. Jeder Slot kann dafür verwendet werden, einen Container oder ein Gerät mit einem Mount-Directory zu verbinden. Doch zunächst braucht man einen Container. Diesen erstellt man durch Klick auf **Create Volume** und Auswahl von „Create an encrypted file container“ im sich darauf öffnenden Fenster. Ich will ein Standard Volume erzeugen und wähle das im nächsten Fenster aus. Alternativ kann man auch Hidden Volume wählen, was einen Container im Container erzeugt. Falls man gezwungen wird, das Passwort für den

äußeren Container herauszugeben, ist der innere nach wie vor verborgen. Wenn im äußeren scheinbar wichtige Daten sind, geben sich diejenigen, die einen zwingen, eventuell zufrieden. Nach Klick auf „Next“ kann man jetzt eine Datei auswählen bzw. neu erstellen lassen, die als Containerdatei dienen soll. Ich erstelle eine Datei namens „truecryptcontainer“ und klicke auf „Next“.

Im nächsten Fenster kann man jetzt den Algorithmus auswählen, den man verwenden will. (Abb. 13)

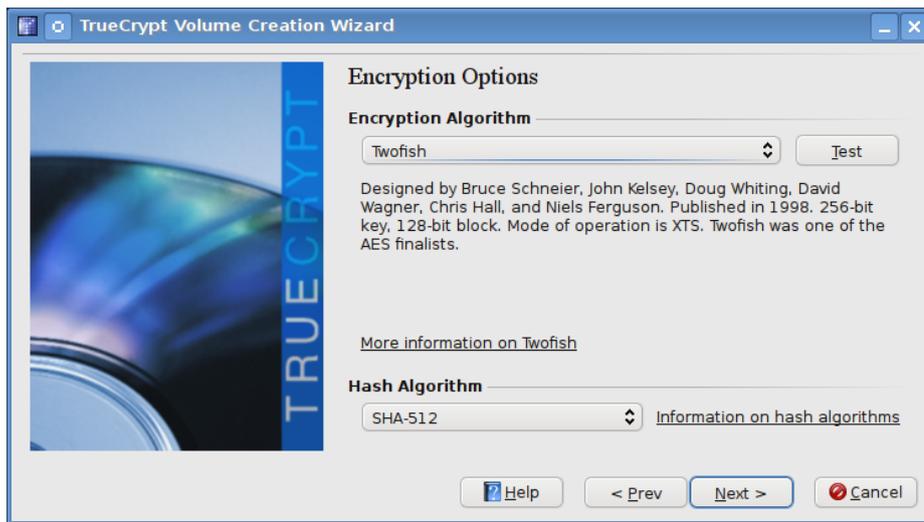


Abbildung 13: Auswahl des Algorithmus.

Für den Unterricht interessant ist, dass hier zusammengefasste Informationen des Algorithmus nachzulesen und auch weiterführende Links zu finden sind. Weiters muss man sich nicht auf einen Algorithmus beschränken, man kann auch zwei oder drei auswählen, die hintereinander ausgeführt werden. Als Algorithmus wähle ich Twofish, als Hash SHA-512 und klicke auf „Next“.

Nun muss man die Größe des Containers angeben, ich wähle 50 MB. Nach dem nächsten Klick auf Next kann man entweder ein Passwort, oder ein Keyfile angeben. Es wird empfohlen, ein Passwort von mindestens 20 Zeichen Länge aus Klein- und Großbuchstaben, Zahlen und Sonderzeichen zu wählen. Wie man im Abschnitt „Passwörter“ auf Seite 92 lesen kann, wollen die TrueCrypt-Entwickler einen zu einem Passwort, das 2^{128} Bits Sicherheit bietet, drängen. Als nächstes kann man das Filesystem wählen. Aus Kompatibilitätsgründen wähle ich FAT. Im nächsten Fenster soll man möglichst lang seine Maus über dem Fenster bewegen, um Zufallsdaten zu sammeln, welche für die Formatierung des Containers benötigt werden. Nach Formatierung kann man mit „Exit“ diesen Dialog verlassen.

Jetzt will ich den gerade erstellten Container ins Dateisystem einbinden, das heißt mounten. Dafür wähle ich den Slot 1 aus und klicke auf „Select File“. (Abb. 12) Nach dem Klick auf „Mount“ muss man zunächst das Passwort für den Container, und anschließend das Root-Passwort eingeben (bzw. das des Users, der die entsprechenden Rechte über sudo hat). Wenn man kein Mount Directory angegeben hat, wird automatisch nach dem Schema `/media/truecryptX` gemountet, wobei X die Slotnummer ist. Alle Daten, die jetzt nach `/media/truecrypt1` kopiert werden, werden mit TwoFish verschlüsselt. Wenn man fertig ist, wählt man den Slot wieder aus und klickt auf „Dismount“. Damit fällt die Tür ins Schloss und die Daten sind sicher.

TrueCrypt ist mit Sicherheit sehr praktisch, wenn zwei oder mehr Menschen größere Daten verschlüsselt austauschen wollen. Durch die grafische Benutzeroberfläche, lässt es sich komfortabel bedienen. Außerdem ist es für jede gängige Plattform verfügbar. Wenn man z.B. Daten auf CD oder DVD gebrannt per Post verschicken will, ist es eine gute Idee, sie in einen verschlüsselten Container zu kopieren und diesen auf die CD oder DVD zu brennen.

3.4 One-Time-Pad

Das One-Time-Pad-Verschlüsselungsverfahren ist so einfach, dass man gar nicht glauben möchte, dass es erst 1917 von Major Joseph Mauborgne und AT&Ts Gilbert Vernam erfunden wurde. Doch nicht nur die intrinsische Einfachheit des OTP-Verfahrens ist charakteristisch. Auch, dass es das einzige Verfahren ist, das nachweislich unknackbar ist, selbst mit unendlichen Ressourcen. Bei jedem anderen Kryptosystem kann man zumindest jeden möglichen Key durchprobieren, eine sogenannte *Brute Force*-Attacke. Bei OTP erhält man aber für jede Folge von Ciphertext-Zeichen jede andere Klartext-Zeichenfolge, die die gleiche Länge hat. Davon sind auch mehr als genug sinnvolle Wörter oder Sätze. [12]

Was man für OTP-Verschlüsselung braucht ist eine Nachricht, einen zufällig generierten Schlüssel, der genau so lang ist wie die Nachricht selbst und zwei Kommunikationspartner. Beide müssen im Besitz dieses Schlüssels sein, was die größte Herausforderung beim OTP darstellt, die sichere Übertragung des Schlüssels. Angenommen ich will mit meinem Kollegen David sicher kommunizieren und wir wollen dies über OTP machen. Dann werde ich ein paar A4-Seiten voll mit zufälligen Symbolen aus dem Alphabet A ... Z, 0 ... 9, Leerzeichen und Punkt ausdrucken. Und zwar zweimal. Einmal für mich und einmal für ihn. Und selbstverständlich ist es beide Mal die selbe Reihenfolge. Eventuell ist auch die Anordnung in einem Raster der

Zeichenblöcke sinnvoll, wie man später vielleicht sehen wird.

3.4.1 Funktionsweise und Anwendung

Um zu verstehen, was passieren wird, müssen wir zunächst wie gewohnt die Zeichen die wir verwenden wollen in Zahlen umrechnen. A bis Z sollen die Zahlen 0 bis 25 haben, 0 bis 9 26 bis 35, Leerzeichen bekommt 36 und der Punkt soll 37 bekommen. Die Zahlen bewegen sich also zwischen 0 und 37, genau der Zahlenbereich, den man erhält, wenn man beliebige Zahlen modulo 38 rechnet. Zunächst wandle ich meine Nachricht in Zahlen um. Die Nachricht wird HALLO DAVID. lauten. Das H wird zu 7, das A zu 0 usw. bis ich beim Punkt angelangt bin, der durch die Zahl 37 ersetzt wird. Die Nachricht lautet somit zunächst 7 0 11 11 14 36 3 0 21 8 3 37. Jetzt geht es ums Verschlüsseln. Dafür brauche ich die ersten zwölf Zeichen aus unserem Schlüssel auf dem Papier. Die Nachricht ist zwölf Zeichen lang, also muss auch der Schlüssel so lang sein. Der Schlüssel soll diese Zahlenfolge sein: 18 9 29 12 12 32 25 22 24 22 27 4. Auf dem Papier steht dort somit SJ3MM6ZWYW1E. Jetzt wird Zahl für Zahl der Nachricht mit ihrem Gegenstück aus dem Schlüssel addiert und modulo 38 genommen. Also die erste Zahl der Nachricht mit der ersten Zahl des Schlüssels usw. Wenn die Summe der Zahlen kleiner als 38 ist, kann man sich das Modulo-Rechnen natürlich ersparen. Somit rechnen wir $7 + 18 = 25$, $0 + 9 = 9$, \dots , $36 + 32 = 30 \bmod 38$, \dots , $37 + 4 = 3 \bmod 38$. Der Ciphertext ist somit schlussendlich 25 9 2 23 26 30 28 22 7 30 30 3. Übersichtlich aufgeschrieben sieht man sieht das dann so aus:

```
N: 7 0 11 11 14 36 3 0 21 8 3 37
K: 18 9 29 12 12 32 25 22 24 22 27 4
C: 25 9 2 23 26 30 28 22 7 30 30 3
```

David erhält dann meine Nachricht auf beliebigem Weg. Da sie ohnehin absolut sicher verschlüsselt wurde, kann man sie auch als Zeitungsinserat aufgeben. Er subtrahiert nun von der Nachricht den Schlüssel und addiert bei negativem Ergebnis 38. Er rechnet somit $25 - 18 = 7$, \dots , $30 - 32 = -2$, $-2 + 38 = 36$, \dots , $3 - 4 = -1$, $-1 + 38 = 37$. Wenn er die Zahlen wieder durch die im Alphabet vereinbarten Zeichen ersetzt, hat er die ursprüngliche Nachricht wieder hergestellt.

Warum einen Raster, wie beim Schiffeversenken, auf den Schlüssel-Bögen anlegen eine gute Idee sein könnte, ist offensichtlich. Die Kommunikation kann nur dann funktionieren, wenn wir beide genau die selbe Sequenz des Schlüssels verwenden. Wenn

wir auch nur ein Zeichen verschoben sind, kommt nichts richtiges heraus. Sagen zu können, durchaus über einen unsicheren Kanal, wir verwenden Zeichen B7 bis R19, ist also eventuell eine gute Idee.

Händisch herumzurechnen ist vielleicht ab und zu ganz witzig und versetzt einen zurück in die Rolle eines Spions zur Zeit des Zweiten Weltkriegs (an dieser Stelle möchte ich Neal Stephenson's „Cryptonomicon“ empfehlen), ist aber auf Dauer recht mühsam. Genau für solche Arbeiten hat man ja elektronische Computer. Das Gute an elektronischen Computern ist, dass ihr Alphabet binär ist und damit ist OTP ganz schnell erledigt. Man braucht nur eine XOR, also Ausschließendes-Oder-Verknüpfung aus Nachricht und Key, bzw. Key und Ciphertext zu machen. Ein Beispiel: Die Nachricht soll $N=01000001$ sein. Der Key ist wieder zufällig gewählt und lautet $K=10100100$. Die Verknüpfungstabelle für XOR sieht wie in Tabelle 9 aus. Als Symbol für XOR wird hier wie im Abschnitt „AES“ (Seite 45) \oplus gewählt.

Tabelle 9: Verknüpfungstabelle für XOR

a	b	a \oplus b
0	0	0
0	1	1
1	0	1
1	1	0

Wieder übersichtlich aufgeschrieben erhält man den Ciphertext C . Wir verknüpfen von oben nach unten \oplus .

$N=01000001$

$K=10100100$

$C=11100101$

C wird wieder übermittelt. David macht jetzt nichts anderes als von unten nach oben \oplus zu verknüpfen. Man sieht schon, dass wieder die ursprüngliche Nachricht herauskommt. N war übrigens 65 was als ASCII-Zeichen interpretiert ein A ist. Ein Programm zu schreiben, dass zwei binäre Strings entgegennimmt und die \oplus -Verknüpfung davon ausgibt lässt sich bestimmt im Informatikunterricht unterbringen.

Das Wichtigste beim OTP ist zu beachten, dass der Schlüssel geheim bleiben muss und dass er zufällig sein muss. Wirklich zufällig, nicht pseudozufällig und dass man ihn nur einmal verwenden darf. Nach Verwendung kann man ihn, und sollte man ihn, also zerstören. Damit ist er für immer weg und kann auch nicht mehr verwendet werden.

Die Sache mit dem Zufall. Für die Schlüssel in den hier verwendeten Beispielen habe ich immer einfach nur in einer Unix-Shell (Bash auf GNU/Linux in meinem Fall) `echo $(($RANDOM % A))` eingegeben, wobei A durch 38 bzw. 2 ersetzt wird. Ob das für wirkliche Kommunikation, die aus beliebig vielen Zeichen besteht auch eine ausreichend gute Idee ist, kann ich schwer sagen. Ich weiß nicht, wie die Zahl in der Umgebungsvariable `$RANDOM` erzeugt wird. Wenn die Quelle `/dev/random` ist, dann sollte die Zufälligkeit echt sein und ausreichen. Wenn sie aus Gründen der Geschwindigkeit pseudozufällig befüllt wird, dann sollte man ihr nicht vertrauen. Zu erkennen, was zufällig ist, und was eine regelmäßige Sequenz ist, deren Regelmäßigkeit man nicht kennt, ist sehr schwer. Deshalb muss die Quelle des Schlüssels absolut vertrauenswürdig sein. Ein Beispiel was der Unterschied zwischen einer zufälligen Sequenz und einer zufällig aussehenden Sequenz ist: LMMJVSD, WWHVHBE. Was davon ist zufällig, und was nicht? Ein bisschen noch nachdenken... Nichts gefunden? Wer in der zweiten Sequenz meint, ein Muster erkannt zu haben, der soll mir das bitte mitteilen. Hier habe ich nämlich wieder auf `$RANDOM` zurückgegriffen. Die erste Sequenz wird aber vielleicht Menschen mit Spanischkenntnissen aufgefallen sein. Es sind einfach die Anfangsbuchstaben von Montag bis Sonntag, nur eben auf Spanisch. Zusammengefasst gesagt ist `/dev/random` wahrscheinlich eine gute Wahl, wenn man Zufallszahlen haben möchte, eine genaue Recherche, wie die Zahlen zustande kommen, ist aber nicht verkehrt. Auch gut sind natürlich alle Geräte die irgendeinen quantenmechanischen Prozess verwenden. Recht einfach müsste eigentlich folgendes zu realisieren sein: Man nimmt einen Strahler, Alpha am besten oder Beta, den müsste man aber entsprechend abschirmen, und platziert gegenüber davon eine Photozelle die bei Auftreffen eines Partikels an der Stelle einen Punkt meldet. Dann teilt man die Zelle logisch in links und rechts ein. Trifft der Partikel auf die linke Seite, wird dies als 0 interpretiert, auf der rechten Seite als 1. Da es nicht vorhersagbar ist, wann ein Atom zerfällt und wohin der Partikel genau fliegen wird, hat man hier eine gute Zufallsquelle. Je nach Aktivität kann es aber recht lang dauern, bis man genug zufällige Werte hat, um damit vernünftig arbeiten zu können.

Was ist mit Würfeln, Münzen werfen, ein Bingo- oder Lotto-Apparat, oder Karten aus einem gemischten Kartendeck ziehen? Ist das nicht auch zufällig? Schon, aber wer würfelt, wirft Münzen oder zieht Bingo-Bälle? Ein Mensch. Und Menschen haben eine Tendenz. Sie glauben immer alles besser zu wissen. Sogar als der Zufall. Wenn also jemand Bingo-Bälle zieht und er zieht zweimal hintereinander B7, dann wird er misstrauisch. Beim nächsten Reingreifen wird er ganz von alleine und automatisch

nicht dort hingreifen, wo er B7 vermutet, sondern wo anders. Vielleicht sieht er B7 sogar und greift absichtlich daneben, obwohl der Ball eigentlich wieder dort liegt, wo er normalerweise hingreift. Denn dreimal hintereinander B7 zu ziehen, das geht doch nicht. Doch, geht. Das passiert. Wenn man Zehntausend mal einen Bingo-Ball zieht und man erhält nicht mindestens einmal eine Dreierfolge, wäre das eher seltsam. Also mit geschlossenen Augen? Ja, das ist eine Verbesserung, aber nicht gut genug. Auch hier wird der Mensch wieder versuchen, tendenziell weg von B7 zu kommen. Vielleicht schmeißt man den B7-Ball wieder zurück und wertet die Ziehung nicht. Oder man wird schlampig und macht die Augen nicht mehr richtig zu. Man öffnet sie zu früh, weil man sich nicht schon wieder an der Türklappe der Trommel stoßen möchte. Oder man lernt die Bälle zu gut kennen. Der B7-Ball hat eine kleine Eindellung, weil er mal heruntergefallen ist, die man spürt, wenn man ihn angreift. So etwas geht automatisch und niemand ist dagegen resistent. Deshalb diese Dinge besser Maschinen überlassen. Ähnliche Szenarien wurden in der Realität bereits ausgenutzt und sind ebenfalls in *Cryptonomicon* so beschrieben. [17]

3.4.2 Unterrichtssequenz

Wie bereits weiter oben erwähnt, ist es leicht möglich in den Informatikunterricht eine Unterrichtssequenz zu machen, in der ein Programm geschrieben wird, das OTP realisiert. Ob das nun auf binärer Ebene passieren soll, wo man nichts anders als zwei Strings XOR-Verknüpft, oder ob man sich auf ein beliebiges andere Alphabet einigt, ist fast egal. Der Aufwand ist bei letzterer Realisierung nur ein wenig größer. Bevor man die Schülerinnen und Schüler jedoch die Programmierung einer OTP-Software beginnen lässt, sollte man sie auch händisch ein paar Wörter ver- und entschlüsseln lassen, damit sie wirklich verstehen, was hier passiert. Auch die Generierung eines Schlüssels kann durchaus einmal händisch passieren, um zu sehen, wie Aufwendig es eigentlich ist, zufällige Folgen zu erhalten. Blind auf die Tastatur hauen ist leider hierfür nicht gut genug. Es müssen schon tatsächlich Münzen geworfen werden oder ähnliches.

3.5 Solitaire

Eines meiner Lieblingsbücher ist *Cryptonomicon* von Neal Stephenson. [17] Ein wesentlicher Bestandteil in der, in viele Handlungsstränge aufgeteilten, auf zwei Epochen verteilten, Geschichte die in diesem Buch erzählt wird, ist Kryptographie und

Kryptoanalyse. Einerseits die ersten Gehversuche im Bereich von kryptographischen Algorithmen während des Zweiten Weltkriegs, gemeinsam mit dem Versuch diese zu brechen, andererseits die wie selbstverständliche Anwendung in der Gegenwart. Ein besonderer von Bruce Schneier entwickelter Algorithmus, der in diesem Buch eine Rolle spielt, soll hier besprochen werden. Es handelt sich um *Solitaire*, im Buch heißt er allerdings „Pontifex“. Schneier entwickelte Solitaire für im Außendienst befindliche Agenten, damit diese sicher kommunizieren können, auch wenn sie keinen Zugang zu dafür benötigten elektronischen Geräte haben, oder wenn der Besitz von solchen Geräten zu riskant wäre. Ein Kartendeck bei sich zu haben sollte wenig Aufmerksamkeit erregen, denn das ist alles, was man für diese Verschlüsselungsmethode benötigt. Zu Nutze macht sich Solitaire die Zufälligkeit der Anordnung von Karten in einem Kartendeck. Das Kartendeck selbst muss alle 52 Karten und zwei verschiedene Joker umfassen, also insgesamt 54 Karten. [17]

3.5.1 Funktionsweise von Solitaire

Solitaire ist ein Key-Stream Cipher. Das heißt, man bedient sich eines Algorithmus den man auf, in diesem Fall, ein Kartendeck anwendet und generiert daraus einen Strom an Zeichen für den Schlüssel. Mit diesen Zeichen verschlüsselt man dann den Klartext, was mittels Addition Modulo 26 geschieht. Diese Form der Addition kommt in der Kryptographie ständig vor. Auf Bit-Ebene ist sie Modulo 2 oder auch XOR genannt. Im Abschnitt *One-Time-Pad* wird ausschließlich darauf zurückgegriffen. Hier wird noch einmal kurz erklärt, wie das funktioniert. Auf jeden Fall muss aber klar sein, dass man für jedes Zeichen aus dem Klartext, ein frisch erzeugtes Zeichen aus dem Key-Stream braucht. Auf keinen Fall reicht es, n Schlüsselzeichen zu generieren und für einen $2n$ langen Text einfach jedes Zeichen des Schlüssels zweimal zu verwenden. Diese oder ähnliche Vereinfachungen schwächen die Verschlüsselung enorm und machen den Aufwand komplett unnötig.

Der Klartext darf nur Buchstaben im Alphabet zwischen A und Z haben. Keine Ziffern, Sonderzeichen, oder Satzzeichen. Außerdem wird er in Blöcke von je fünf Zeichen unterteilt und in Großbuchstaben notiert. Blöcke, die weniger als fünf Zeichen lang sind, werden mit X aufgefüllt. Das alles ist selbstverständlich reine Konvention und kann je nach Bedürfnis verändert werden. Wichtig ist nur, dass beide Kommunikationspartner das gleiche machen, also warum nicht gleich die Konvention aus Cryptonomicon übernehmen?

Hier soll ein kleines Beispiel demonstriert werden wie die Verschlüsselung mit den

bereits generierten Zeichen des Keys abläuft.

Verschlüsselung Zunächst muss der Klartext in Blöcke von je fünf Zeichen unterteilt werden. Hier wird der Text „HALLO BOB“ verschlüsselt. Das ergibt somit die Blöcke HALLO BOBXX. Wie man sieht, wurden die letzten beiden Zeichen des nur zu 3/5 gefüllten zweiten Blocks mit X aufgefüllt. Das gleiche machen wir jetzt mit dem Key, aber natürlich wird hier nicht mit X aufgefüllt, sondern auch für die Dummy-X werden vernünftige Schlüsselzeichen generiert. Hier nehme ich die zufällig ausgewählten Schlüsselzeichen JKBFN BLDMB. Jetzt müssen die Buchstaben in Zahlen umgewandelt werden. Auch hier ist wieder Konvention, dass A mit 1 ersetzt wird, B mit 2 usw. Der Klartext von vorhin wird somit zu 8 1 12 12 15 2 15 2 24 24. Der Schlüssel wird zu 10 11 2 6 14 2 12 4 13 2. Jetzt wird das erste Zeichen des Klartexts mit dem ersten Zeichen des Schlüsselstroms addiert. Wenn das Ergebnis größer als 26 ist, wird 26 subtrahiert. Das ergibt somit 18 12 14 18 3 4 1 6 11 26. Diese Zahlen kann man jetzt wieder zurück in Buchstaben umwandeln, womit man den Ciphertext RLNRC DAFKZ erhält.

Entschlüsselung Natürlich müssen beide miteinander kommunizierende Parteien den gleichen Schlüssel haben um den gleichen Schlüsselzeichen-Strom zu erzeugen. Der Empfänger wandelt zunächst den erhaltenen Ciphertext wieder in Zahlen um und generiert so viele Zeichen des Schlüsselstroms, wie er braucht. In diesem Fall sind es zehn. Danach subtrahiert er die Zahlen des Schlüsselstroms von denen des Ciphertexts, wieder Modulo 26. Das heißt, dass wenn das Ergebnis kleiner oder gleich 0 ist, wird 26 addiert. Man sieht sofort, dass wieder der Klartext HALLO BOBXX zum Vorschein kommt.

Schlüsselstrom-Zeichen erzeugen Was bisher geschah, war Verschlüsselungs-Ein-Mal-Eins. An dieser Stelle soll erklärt werden, wo sich die eigentliche „Magie“ von Solitaire abspielt, nämlich in der Erzeugung der Schlüsselstrom-Zeichen.

Das Deck hat, wie gesagt 52 Karten und zwei verschiedene Joker. Ich verwende gerade eines, in dem ein Joker rot und einer schwarz ist. In den meisten Spielen unterscheiden sich die Joker irgendwie voneinander. Jedenfalls braucht man aber einen A-Joker und einen B-Joker. Ich nenne den roten den A-Joker. Sollten sich die Joker kurioser Weise nicht voneinander unterscheiden, oder hat man Probleme damit, sich rot=A und schwarz=B zu merken, dann kann man natürlich auch A oder B auf die Joker schreiben. Man sollte aber eine gute Ausrede dafür parat haben.

Nun geht es daran, das Deck zu *initialisieren*, das heißt, dafür bereit zu machen, den Keystream zu generieren. Dafür nimmt man das Deck mit den Bildern nach oben in die Hand und bringt die Karten in die Anfangsreihenfolge, die den Schlüssel darstellt. Der Schlüssel ist *nicht* der Schlüsselstrom! Wie man das macht, wird im Abschnitt „Keying the Deck“ erklärt.

So wird ein ein Zeichen produziert:

1: Finde den A-Joker. Bewege ihn eine Karte hinunter, das heißt, tausche ihn mit der Karte, die unter ihm liegt aus. Ist der Joker die letzte Karte im Deck, dann stecke ihn unter die oberste Karte. (Man kann sich vorstellen, dass die Karten in einem Ring angeordnet sind. Dabei „liegt“ die im Stapel ganz unterste Karte auf der obersten.)

2: Finde den B-Joker und bewege ihn zwei Karten hinunter.

Wichtig ist, diese zwei Schritte nacheinander zu machen und nicht das Kartendeck durchzugehen und die Joker in der Reihenfolge zu verschieben, wie man sie antrifft. Erst den A-Joker, dann den B-Joker.

3: Triple Cut Jetzt muss man ein dreigeteiltes Abheben vollziehen. Das heißt, dass man die Karten die *über* dem ersten Joker sind mit denen *unter* dem zweiten tauscht. Dabei bedeutet jetzt „erster“ und „zweiter“ Joker nur, welcher im Stapel weiter oben und welcher weiter unten ist. Die Joker selbst und die Karten zwischen ihnen bewegen sich nicht! Wenn der Sub-Stapel „Über“ bzw. „Unter“ leer ist, dann wird er trotzdem „bewegt“. Sollte sowohl die oberste, als auch die unterste Karte ein Joker sein, werden zwei leere Sub-Stapel bewegt, das heißt, es passiert nichts.

4: Count Cut Jetzt macht man einen „Count Cut“. Das heißt, man sieht sich die unterste Karte an und wandelt sie in eine Nummer zwischen 1 und 53 um. Dabei verwenden wir die **Bridge-Wertigkeiten: Kreuz, Karo, Herz und Pik**. Außerdem soll Ass den Wert 1 haben, 2 den Wert 2 usw. (Bei einer Kreuz-Karte, ist der Wert das, was auf der Karte steht. Bei Karo ist es Wert + 13, bei Herz Wert + 26 und bei Pik Wert + 39. Jeder Joker hat den Wert 53.)

Dann zählt man von der obersten Karte an die Karten bis man zur Karte an der Stelle kommt, die dem Wert entspricht. Zum Beispiel war bei mir gerade die

unterste Karte die Herz-Dame. Das entspricht der Wertigkeit 38. Ich zähle also bis zur 38. Karte hinunter. Dann nimmt man den Sub-Stapel zwischen der 38. Karte und der untersten und legt ihn oben auf. Die 38. Karte und die unterste sind *nicht* Teil dieses Sub-Stapels. Am Ende dieses Schrittes, müsste die 38. Karte über der untersten liegen. Wenn ein Joker die unterste Karte ist, passiert in diesem Schritt nichts.

5: Output-Karte finden. Nun wandelt man die oberste Karte im Deck, wie in Schritt 4, in eine Nummer um. Bei mir ist es Karo 4, das bedeutet 17. Jetzt zählt man wieder so viele Karten hinunter. Die Karte *unter* der, zu der man jetzt gezählt hat, schreibt man auf ein Blatt Papier. Dies ist die erste Output-Karte. Bei einem Joker schreibt man nichts auf und beginnt wieder bei Schritt 1. Bei diesem Schritt bleibt das Deck unverändert!

6: Output-Karte zu Nummer Jetzt muss die Output-Karte nur noch in eine Nummer umgewandelt werden, so wie vorhin auch.

So kann man sich so viele Zeichen aus dem Keystream erzeugen, wie man braucht, man geht einfach Schritt 1 bis 6 wieder und wieder durch, ohne aber das Deck dem Schlüssel entsprechend neu anzuordnen. Das heißt, nach Schritt 6 wieder Schritt 1 ausführen, und nicht neu initialisieren.

Ein Tipp von Bruce Schneier an dieser Stelle: Wenn man mit hohen Karten arbeitet, also zum Beispiel Herz Bube, was 37 entspricht, dann zählt er nicht von 1 bis 37 sondern zwei mal bis 13 und ein mal bis 11.

Keying the Deck: Solitaire ist ein Algorithmus, der getreu dem Prinzip *Security by design* entwickelt wurde. Das heißt, der Algorithmus ist für jeden zugänglich und analysierbar und die Stärke der Verschlüsselung hängt im Großen und Ganzen ausschließlich von der Stärke und Geheimhaltung des verwendeten Schlüssels ab. Wenn zwei Parteien mit Solitaire verschlüsselt kommunizieren wollen, müssen sie sich klarer Weise auf den gleichen Schlüssel einigen, um den gleichen Schlüsselstrom zu erzeugen.

Der stärkste Schlüssel bei Solitaire ist das ganze Deck gut zu mischen und ein zweites, identisches Deck für den Kommunikationspartner zu machen. Beide sollten darüber hinaus dann auch noch jeweils ein weiteres Deck mit der selben Reihenfolge der Karten haben, damit sie bei einem Fehler wieder auf Anfang zurück können. Kann man nach einem (oder mehreren) Fehlern den richtigen Zustand des Decks nicht mehr herstellen, ist die Kommunikation unmöglich. Herr Schneier empfiehlt das Deck

mindestens sechs mal zu mischen, da die meisten Menschen schlechte Mischer sind. Ob man Mischautomaten vertrauen kann, sei an dieser Stelle als fraglich markiert.

In manchen Zeitungen gibt es eine Auflistung von Bridge-Spielhänden. Man kann sich mit seinem Kommunikationspartner eine bestimmte Zeitung ausmachen und dann immer die aktuelle Bridge-Hand des Tages zur Verschlüsselung verwenden. Das ist aber nicht unbedingt sicher, denn auch potentielle Abhörer wissen um diese Taktik Bescheid und müssen eventuell nur ein paar Bridge-Hände durchprobieren, denn wie viele Zeitungen, in denen Bridge-Konstellationen abgebildet sind und zu denen beide Partner Zugang haben, gibt es? Das kommt wohl darauf an, wo sie sich befinden.

Die dritte Möglichkeit ist, eine Passphrase zu verwenden. Zum Beispiel einigt man sich beim letzten persönlichen Treffen, oder über einen anderen sicheren Kanal, auf die Passphrase **GEHEIMESPASSWORT**. Wenn dann beide wieder getrennter Wege gehen und sich mittels Solitaire unterhalten wollen, nehmen sie ein Deck und bringen es in eine bestimmte, auch vorhin definierte, Position. Zum Beispiel die Karten in aufsteigender Reihenfolge nach Bridge-Wertigkeit und am Ende den A-Joker und B-Joker. Dann macht man den Algorithmus bis inklusive Schritt 4 und anschließend nochmal Schritt 4 aber diesmal zählt man den ersten Buchstaben der Passphrase hinunter. In diesem Beispiel G was 7 entspricht. Wieder den Sub-Stapel zwischen 7. und letzter Karte oben auflegen und bei Schritt 1 weitermachen. Schritt 5 und 6 werden hier nicht gebraucht. Beim zweiten Durchgang wird selbstverständlich das zweite Zeichen der Passphrase verwendet usw. Als optionalen Schritt nennt Schneier noch, die Joker wie folgt anzuordnen: Den A-Joker hinter die R-te Karte stecken und den B-Joker hinter die T-te, also vorletztes und letztes Zeichen der Passphrase als Position für die Joker zu verwenden. [17]

Dass die Passphrase am besten möglichst lang und möglichst zufällig sein sollte, muss wahrscheinlich nicht extra erwähnt werden. Im Standard-Englisch ist pro Buchstabe nur eine Zufälligkeit von 1,4 gegeben. Deshalb empfiehlt Schneier eine Passphrase von mindestens 64 Zeichen, besser aber 80.

Auf <http://www.schneier.com/solitaire.html> Sind noch weitere Tipps im Umgang mit Solitaire und Passwörtern. Außerdem findet man dort den Algorithmus in diversen Programmiersprachen implementiert, was zumindest dem zuhause gebliebenen Agenten freut, weil der somit nicht per Hand ver- und entschlüsseln muss.

Ich bin den Algorithmus ein paar mal durchgegangen. Nach drei bis vier Durchgängen hat man den Dreh raus und ist recht flott mit dem Abzählen und Umordnen der Karten. Worauf ich achten muss, und eventuell auch andere, ist, dass ich oft leise mit-

zähle, was man eventuell abhören könnte. Das heißt, man sollte darauf achten, alles im Kopf durchzuzählen. Außerdem sollte man sicher gehen, dass man nicht beobachtet wird, da man sicher relativ leicht die Schritte rekonstruieren kann, wenn jemand mitschreibt, wie oft man die *Karte mit Daumen verschieb*-Bewegung gemacht hat.

3.5.2 Unterrichtssequenz

Solitaire ist eines der wenigen Kapitel der Informatik, bei dem es sich förmlich aufdrängt, fast handwerklich zu arbeiten. Viele Menschen setzen Informatik mit „Arbeit am Computer“ gleich, was aber eine Verzerrung der Realität ist. Informatiker bedienen sich nur recht häufig des Computers, weil der die formalen Konzepte recht gut umsetzen kann. Der niederländische Informatiker Edsger Wybe Dijkstra¹³ sagte einst: „Computer Science is no more about computers than astronomy is about telescopes.“

Im Unterricht könnte man dann Solitaire einerseits in einer Programmiersprache begutachten, wobei man Variablen keine verräterischen Namen wie „Topcard“ oder „Ajoker“ gibt. Die Schülerinnen und Schüler sollen dann versuchen, zu verstehen, was der Algorithmus macht. Im Anschluss kann man dann vorführen, wie man ihn mit Hilfe von Karten durchführt und, dass die beiden Varianten im Prinzip äquivalent sind. Es macht keinen Unterschied, ob man den Algorithmus von einem Computer, einem trainierten Affen, oder einem Menschen durchgehen lässt.

¹³Kein Tippfehler.

4 Quantenkryptographie

Wie im Abschnitt „One-Time-Pad“ bereits erwähnt, ist jeder kryptographischer Algorithmus, außer OTP, knackbar, wenn man genug Ressourcen dafür hat. Es ist nur eine Frage des Geldes und der zur Zeit verfügbaren Rechenleistung. Um abzuschätzen, wie lange ein mit Verfahren X verschlüsselter Text sicher ist, muss man schätzen können, wie schnell die Rechenleistung in Zukunft zunehmen wird. Pessimisten, oder Menschen deren Leben davon abhängt, werden dann eine konservative Abschätzung machen und annehmen, dass sobald sie ihren Ciphertext rausgeschickt haben, begonnen wird ihn mit dem besten Supercomputer der Welt zu entschlüsseln. Unter der Annahme, dass auch der Supercomputer mit der Zeit dank technologischer Fortschritten besser wird, kommt er dann auf einen Zeitpunkt, zu dem er, im Falle eines Spions, besser nicht mehr hinter feindlichen Linien ist. Und das setzt voraus, dass der Algorithmus keine Schwachstellen hat. Wenn er anfällig auf diverse Heuristiken der Kryptoanalytiker ist, dann kann er nochmal ein paar Jahre abziehen. Wenn dann plötzlich ein kluger Mensch eines Morgens aufwacht und weiß, wie er einen Quantencomputer bauen kann, der sich nicht bei der Rechnung $3 + 4$ plagt, dann kann die Abschätzung von vorhin gleich vergessen werden, dann ist nämlich jeder Ciphertext der mit herkömmlichen Algorithmen erstellt wurde praktisch im Plaintext mitlesbar. Außer natürlich OTP, das bleibt sicher.

Wenn an dieser Stelle nochmal OTP erwähnt wird, dann muss Quantenkryptographie damit etwas zu tun haben. Das Wort „Quantenkryptographie“ ist eigentlich ein bisschen irreführend. Der kryptographische Aspekt davon ist nämlich klassisch ein OTP. Wo die Quanten ins Spiel kommen ist lediglich bei der Keydistribution, also dem Verteilen, bzw. dem Ausmachen des Keys zwischen den kommunizierenden Parteien. Denn nehmen wir an, wir hätten schon Quantencomputer und RSA und die anderen tollen Algorithmen sind wertlos. Wie soll man über große Distanzen sicher kommunizieren? Ein Public Key-Verfahren funktioniert nicht, RSA ist ja kaputt. Und sich persönlich treffen und dabei Gigabytes an One-Time-Pads auszutauschen ist auch eher unpraktikabel. Bei Quantumkeydistribution, ich benenne Quantenkryptographie damit um, macht man sich grundlegende, unumgängliche physikalische Eigenheiten zu Nutze, um sicher Schlüssel auszutauschen. Sicher deshalb, weil sie zum einen wirklich zufällig sind und zum anderen weil die kommunizierenden Parteien erkennen können, wenn jemand den Schlüsselaustausch belauscht um an den Schlüssel heranzukommen. Um zu verstehen, wie das funktioniert, muss man ein wenig ausholen und in die

Quantenmechanik eintauchen.

4.1 Quantenmechanik

Hier sollen Eigenheiten der Natur erklärt werden, bei denen man langsam aber sicher an einen Punkt gerät, bei dem die Frage „Und warum ist das so?“ mit einem Satz beginnen muss, der mit „Ich glaube. . .“ beginnt. Es verstehen zu wollen, dürfte nahezu unmöglich sein, denn selbst Richard Feynman, einer der bekanntesten Physiker im Bereich der Quantenfeldtheorie und Nobelpreisträger, meinte:

I think I can safely say that nobody understands quantum mechanics.¹⁴

Die Beschreibung *wie* die Natur ist muss also ohne Begründung *warum* sie so ist bleiben. Und selbst das *Wie* ist ein Modell, eine Annäherung an die Realität und nicht die Realität selbst. Irgendwie muss man den Dingen Namen geben, damit man sie beschreiben und damit arbeiten kann.

Wenn man sich im Universum umsieht, dann wird man zwei Dinge (aus Mangel eines besseren Wortes) finden, die sich fortbewegen. Erstens Wellen und zweitens Teilchen. Beide haben in ihrer Fortbewegung ein bestimmtes Verhalten die für die jeweilige Art charakteristisch ist. Eine Welle, zum Beispiel an der Oberfläche von Wasser, kann an Kanten gebeugt werden. Das heißt, dass sie sich in den geometrischen Schattenraum eines Hindernisses hinein ausbreiten können. Wenn zum Beispiel eine ebene Wasserwellenfront auf eine Mauer trifft, die einen Spalt hat, dessen Breite klein gegenüber der Wellenlänge ist, so tritt hinter der Mauer an der Stelle des Spalts eine kreisförmige Welle hervor (Abb. 14¹⁵).

4.1.1 Wellenphänomene

Wellen können auch miteinander interferieren. Wenn die Mauer von vornhin zwei Spalten hat, dann sind beide Spalte Quellen kreisförmige Wellen. Wenn die Wellenberge bzw. Täler sich überlagern, kommt es zu konstruktiven bzw. destruktiven Interferenzen, das heißt, an manchen Stellen, wird die Amplitude der Welle größer, an manchen kleiner. Im Extremfall wird die Amplitude zum Vierfachen der ursprünglichen, bei maximaler konstruktiver Interferenz, bzw 0, bei maximaler destruktiver Interferenz. Die Amplitude ist die Höhe des Wellenbergs (bzw. die Tiefe des Wellentals).

¹⁴The Character of Physical Law (1965) Ch. 6; auch zitiert in The New Quantum Universe (2003) von Tony Hey and Patrick Walters

¹⁵<http://de.wikipedia.org/w/index.php?title=Datei:Beugungsspalt.svg>

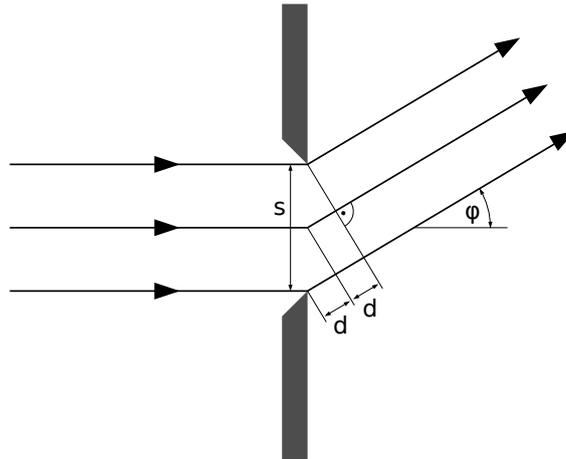


Abbildung 14: Beugung am Einfachspalt

Bringt man in einigem Abstand hinter der Mauer mit den beiden Spalten einen Schirm an, der aufzeichnet, an welchen Stellen beim Auftreffen am Schirm die Wellen wie hoch sind, erkennt man Interferenzmuster, an denen man sieht, wo die Wellen sich maximal verstärkten und wo auslöschten (Abb. 15¹⁶).

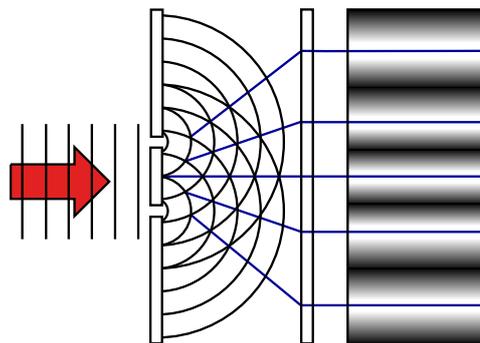


Abbildung 15: Doppelspaltexperiment. Die von den Spalten ausgehenden zylindrischen Wellen überlagern sich und erzeugen am weit entfernten Schirm ein Interferenzmuster ([18], S. 1117).

Diese Phänomene von Beugung und Interferenz scheint einzig und allein bei Wellen angetroffen werden zu können. Wenn man Teilchen bei ihrer Fortbewegung beobachtet, so bewegen sie sich geradlinig weiter, sofern die Summe aller auf sie einwirkenden

¹⁶Leicht modifizierte Version von http://commons.wikimedia.org/wiki/File:Two-Slit_Experiment_Light.svg

Kräfte Null ist. Teilchen können bei Zusammenstößen untereinander oder an einem Hindernis niemals miteinander interferieren, sondern sie ändern ihre Richtung und bewegen sich danach geradlinig weiter.

Ebenso ist der Austausch von Energie unterschiedlich. Während Teilchen ihre Energie bei Zusammenstößen an bestimmten Punkten im Raum zu bestimmten Zeitpunkten austauschen, *breitet sich die Energie von Wellen im Raum aus und wird kontinuierlich übertragen, wenn die Wellenfronten mit Materie wechselwirken.* ([18], S. 1116)

Dass Schall und Licht bzw. elektromagnetische Strahlung generell nichts anderes als Wellen sind, dachte man noch zu Beginn des 20. Jahrhunderts. Diesen Wellen standen Elektronen, Protonen, Neutronen und Atome gegenüber, die man zweifelsfrei für Teilchen hielt. Doch diese strikte Einteilung sollte durch theoretische Überlegungen und neue experimentelle Ergebnisse aufgeweicht werden. Was man bisher wusste ist, dass Licht, wie die im vorherigen Beispiel beschriebenen Wasserwellen, Interferenzmuster an einem Doppelspalt zeigt. Somit war eindeutig, dass Licht Wellencharakter hat. Ein Versuch sollte jedoch zeigen, dass die Lichtenergie nur in bestimmten Portionen auftritt, dass es *quantisiert* ist. Dieser Versuch ist der *fotoelektrische Effekt*.

4.1.2 Der fotoelektrische Effekt, Teilchenphänomene

Bei diesem Versuch (Abb. 16) wird eine Metallplatte mit Licht bestrahlt. Durch das Licht werden Elektronen aus dem Metall der Kathode herausgelöst die dann in Richtung Anode fliegen und dort auftreffen. Dadurch, dass man an der Anode eine negative Spannung anlegt, die dann von den Elektronen überwunden werden muss, kann man herausfinden, mit welcher Energie die Elektronen aus der Kathode herausgelöst wurden.

Das Überraschende war nun, dass egal wie hoch man die Intensität des einfallenden Lichtes werden ließ, sich an der Energie mit der die Elektronen aus dem Metall gelöst wurden, nichts änderte. Klassisch hätte man sich erwartet, dass je höher die Intensität ist, also je größer die Amplitude, desto stärker müssten die Elektronen herausgeschossen werden. Vergleichbar wäre dies mit einem Menschen, der am Strand auch umso stärker von einer Welle umgestoßen wird, je höher ihre Amplitude ist. Bringt man zwischen die Lichtquelle und der Kathode eine Glasscheibe an, werden keine Elektronen mehr aus der Kathode herausgelöst. Das liegt daran, dass das Glas die Anteile im Licht mit höherer Wellenlänge herausfiltert. Die Energie, mit der die Elektronen herausgelöst werden, hängt somit nicht von der Amplitude, sondern von

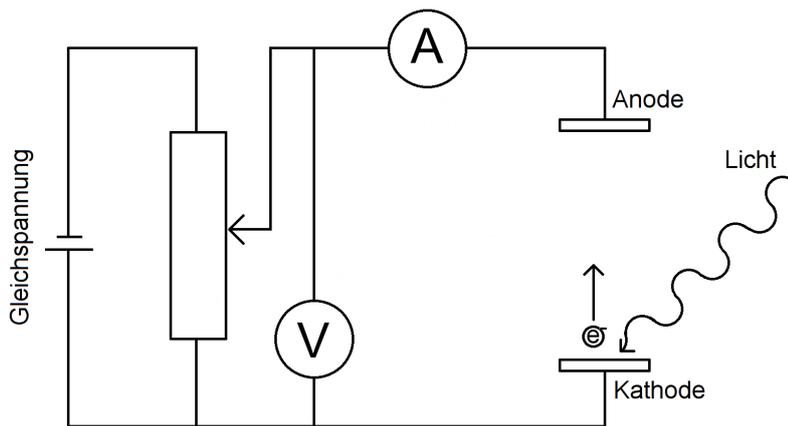


Abbildung 16: Fotoelektrischer Effekt, Versuchsaufbau.

der Frequenz des Lichts ab. Einstein erhielt dann im Jahre 1921 den Nobelpreis für Physik, für die nach ihm benannte Einstein'sche Gleichung

$$E = h\nu = \frac{hc}{\lambda} \quad (30)$$

wobei ν die Frequenz, h das Planck'sche Wirkungsquantum, λ die Wellenlänge und c die Lichtgeschwindigkeit ist. Die Lichtenergie ist also quantisiert, tritt somit in kleinen Paketen auf, die man *Photonen* nennt. [18] Ein Photon muss also eine Energie haben, die hoch genug ist, um ein Elektron herauszulösen. Hat das Photon ein bisschen mehr Energie, werden deshalb nicht eineinhalb Elektronen herausgelöst. So ein Photon ist also ein Teilchen. Einerseits hat man nun die Phänomene von vorhin, die sich nur dadurch erklären lassen, dass Licht eine Welle ist, andererseits sieht man am fotoelektrischen Effekt, dass Licht auch ein Teilchen ist. Diese Formulierung muss man somit verwerfen und sich korrekter ausdrücken: Licht hat sowohl Wellen- als auch Teilchencharakter. Je nachdem, welchen Aspekt man misst, zeigt es den einen, oder den anderen.

Dass die Energie der Photonen mit denen der herausgelösten Elektronen übereinstimmen muss, wird in Abb. 17¹⁷ symbolisch dargestellt.

¹⁷Quelle: http://de.wikipedia.org/w/index.php?title=Datei:Fotoelektrischer_Effekt.svg&filetimestamp=20090504084142

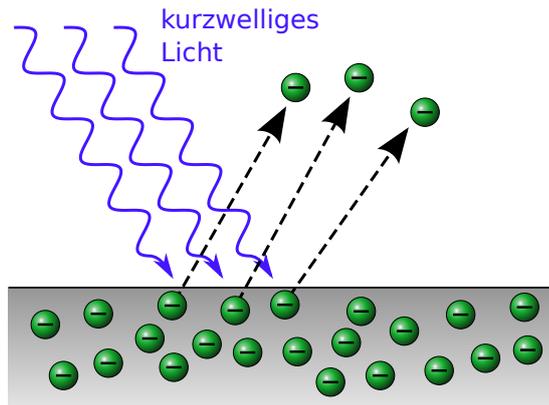


Abbildung 17: Fotoelektrischer Effekt

4.1.3 Teilchen als Wellen

Nachdem man erkannt hat, dass Licht auch Teilchencharakter hat, liegt die Annahme, dass auch Teilchen Wellencharakter haben nicht all zu fern. Der französische Physiker Louis de Broglie postulierte 1924 für die Wellenlänge von Elektronenwellen folgenden Ausdruck:

$$\lambda = \frac{h}{p}. \quad (31)$$

Dabei ist p der Impuls des Elektrons. Dieser Ausdruck sollte für jegliche Materie gelten. Bei makroskopischen Objekten ist demnach aber die Wellenlänge so klein, dass man keine Welleneigenschaften beobachten kann. Dass Elektronen ebenfalls interferieren und gebeugt werden können, wurde eher zufällig von Clinton Joseph Davisson und Lester Halbert Germer bei den Bell Telephone Laboratories entdeckt.

Wenn Elektronen Welleneigenschaften aufweisen, sollte es möglich sein, stehende Elektronenwellen zu erzeugen. Ist die Energie mit der Frequenz einer stehenden Welle assoziiert, etwa über $E = h\nu$, dann wird deutlich, dass stehende Wellen und Energiequantisierung miteinander zusammenhängen. ([18], S. 1125)

4.1.4 Schrödinger-Gleichung

Die Annahme, dass die diskreten Energieniveaus in Atomen durch stehende Wellen zu erklären sind, führte ab 1926 zur Entwicklung der Quantenmechanik durch Erwin Schrödinger und andere. Dabei wird das Elektron mit Hilfe einer Wellenfunkti-

on beschrieben, die einer Wellengleichung gehorcht, der so genannten Schrödinger-Gleichung. Ihre Form hängt beim jeweiligen System von den Kräften ab, die auf das Teilchen wirken. Die Kräfte werden ihrerseits durch die potenzielle Energie beschrieben, die mit diesen Kräften verknüpft ist. Schrödinger löste das Problem stehender Wellen für das Wasserstoffatom, für den einfachen harmonischen Oszillator und auch für andere wichtige Systeme. Die erlaubten Frequenzen führen zusammen mit der Beziehung $E = h\nu$ zu einem Satz von Energieniveaus, der beim Wasserstoffatom experimentell bestätigt wurde. Damit konnte demonstriert werden, dass die Quantenmechanik es ermöglicht, die quantisierten Energieniveaus des jeweiligen Systems zu berechnen. Die Quantenmechanik ist die Grundlage unseres heutigen Verständnisses der Strukturen und Prozesse, von den Vorgängen im Atomkern bis hin zu den Strahlungsspektren ferner Galaxien. ([18], S. 1125)

Nun muss man sich überlegen, wie man die Wellenfunktion eines Elektrons interpretiert. Die Wellenfunktion für Wellen auf einer Saite ist die Saitenauslenkung y in Abhängigkeit von Ort und Zeit. Die Wellenfunktion für Schallwellen kann entsprechend die Auslenkung s eines Luftmoleküls oder der Druck P sein. ([18], S. 1126)

Die Wellenfunktion für Elektronenwellen wird mit dem griechischen Buchstaben ψ symbolisiert. Bei klassischen Wellen, wie denen von Schall und Licht, ist die Energie pro Volumseinheit in der Welle proportional zum Quadrat der Wellenfunktion. ([18], S. 1126) Da die Energie einer Lichtwelle aber quantisiert ist, muss die Energie pro Volumseinheit proportional zur Anzahl der Photonen pro Volumseinheit sein. Wenn man sich die Lichtquelle jetzt so vorstellt, dass die Intensität des Lichts so gering wird, dass nur noch einzelne Photonen ausgesendet werden, dann ist das Quadrat der Wellenfunktion die *Wahrscheinlichkeit*, in irgendeiner Volumseinheit ein Photon anzutreffen.

Da die Schrödinger-Gleichung ein einzelnes Teilchen beschreibt, muss das Quadrat der Wellenfunktion für ein Teilchen die *Wahrscheinlichkeit* angeben, das Teilchen an einem bestimmten Ort zu finden. Weiters muss die Wahrscheinlichkeit proportional zum Volumen dV sein, das Teilchen im betreffenden Volumselement anzutreffen. Die *Wahrscheinlichkeitsdichte* bzw. *Aufenthaltswahrscheinlichkeit* $P(x)$ ist gegeben durch

$$P(x) = \psi^2(x). \quad (32)$$

Die Wahrscheinlichkeit, das Teilchen an einem Punkt x_1 oder x_2 anzutreffen ist die Summe beider Wahrscheinlichkeiten, $P(x_1)dx + P(x_2)dx$. Für den ganzen Raum muss die Wahrscheinlichkeit es irgendwo anzutreffen natürlich 1 sein. Somit ergibt

sich die Normierungsbedingung

$$\int_{-\infty}^{\infty} \psi^2 dx = 1. \quad (33)$$

4.2 Heisenberg'sche Unschärferelation

Die Heisenberg'sche Unschärferelation, benannt nach ihrem Entdecker Werner Heisenberg, besagt, dass es unmöglich ist, alle messbaren Eigenschaften von Quantenteilchen gleichzeitig beliebig genau zu messen. Zum Beispiel hat ein Teilchen einen Impuls und einen Ort. Der Impuls ergibt sich aus der Masse und der Geschwindigkeit des Teilchens. In der klassischen Physik ist es kein Problem, beide Größen beliebig genau zu messen. Zum Beispiel kann man einen Tennisball wiegen, ihn werfen, dabei seine Geschwindigkeit bestimmen und gleichzeitig seine Position messen. Wie gesagt, bei Quantenteilchen ist das nicht mehr möglich und dies wird mathematisch in der berühmten Formel

$$\Delta x \Delta p \geq \frac{1}{2} \hbar \quad (34)$$

ausgedrückt.

x und p sind Ort und Impuls, und \hbar ist $h/(2\pi)$ wobei h wieder das Planck'sche Wirkungsquantum ist. Warum das so sein muss, folgt aus dem Welle-Teilchen-Dualismus. Man kann sich das ungefähr so vorstellen: Wenn man die Position von einem Teilchen messen will, dann ist das am wenigsten invasive was man machen kann, es mit Licht zu bestrahlen und dann aus der Richtung des gestreuten Lichts zu berechnen, wo sich das Teilchen befindet. Je größer die Wellenlänge dieses Lichts ist, desto ungenauer ist die Messung des Ortes. Will man den Ort genauer bestimmen, muss man die Wellenlänge verringern, was aber den Impuls der „Mess-Photonen“ vergrößert und somit den Impuls des Teilchens, dessen Ort man wissen wollte, stärker stört.

4.3 Photonen

Jedes dieser Quantenteilchen kann bestimmte Eigenschaften haben. Bei Photonen ist es so, dass sie eine *Polarisation* haben können. Das heißt, dass das elektromagnetische Feld, durch das Photonen reisen, eine bestimmte Ausrichtung hat. Messen kann man diese Polarisation mit Polfiltern. Polaroidsonnenbrillen sind ein Beispiel dafür. Die Polarisation wird dabei immer in Bezug zu einer Basis bestimmt. Zum Beispiel kann man die Basis HV wählen, was für horizontal und vertikal steht. (Symbol +) Oder

auch die Basis ± 45 , was für $+45^\circ$ und -45° steht. (Symbol \times) Bildlich kann man sich vorstellen, dass das Photon in eine Richtung oszilliert und nur dann durch den Schlitz der z.B. horizontal ausgerichtet ist durchpasst, wenn es auch horizontal oszilliert. Diese Messungen mit den Polfiltern funktioniert so, dass man eine Lichtquelle hernimmt, von der wir die Intensität wissen. Dahinter platziert man dann einen Polfilter, z.B. „vertikal“, oder \updownarrow . Das Photon kommt also unpolarisiert an den Filter an und wird jetzt mit einer Wahrscheinlichkeit von 50% durchkommen, oder nicht. Zwei Dinge sind jetzt wichtig. Jedes Photon das durchkam, ist danach vertikal polarisiert. Vorher war es aber *nicht* vertikal polarisiert. Auch nicht horizontal. Sondern eben gar nicht. Erst im Augenblick der Messung fällt es in den einen, oder den anderen Zustand, wird also absorbiert oder kommt durch. Und dies geschieht absolut zufällig und unvorhersagbar. Platziert man hinter den \updownarrow -Filter einen \leftrightarrow -Filter (also horizontalen), passieren 0% der Photonen. Nimmt man \nearrow -Filter ($+45^\circ$), passieren wieder 50% der Photonen diesen Filter, was also 25% der ursprünglichen Intensität ist. Die Wahrscheinlichkeit, dass Photonen die in einem gewissen Winkel polarisiert sind, durch einen Polfilter im Winkel α passieren beträgt $\cos^2\alpha$. Bei 45° ist das 0,5 oder 50%. Die Quizfrage ist, wie viele Photonen kommen durch, wenn man hinter den \nearrow -Filter einen \leftrightarrow -Filter einbaut. Der Aufbau ist jetzt also $\updownarrow \nearrow \leftrightarrow$. Intuitiv könnte man meinen, dass keines mehr durchkommt. Die Photonen wurden ja bereits vertikal polarisiert, somit ist die Wahrscheinlichkeit, dass sie horizontal passieren 0, da $\cos^2 90 = 0$. Tatsächlich kommen aber 12,5% der ursprünglichen Photonen durch, also wieder die Hälfte. Photonen „vergessen“ also ihre früheren Polarisationen und fangen bei Filtern anderer Basis immer wieder von vorne an. [6]

4.4 BB84-Protokoll

Das BB84-Protokoll zur Übertragung von Schlüsseln macht sich diese Eigenschaften von Photonen zu Nutze. Was passiert, wenn Alice und Bob eine verschlüsselte Nachricht mittels BB84 übertragen wollen? Alice will Bob etwas mitteilen. Als erstes ruft Alice Bob an und sagt ihm, dass sie ihm etwas wichtiges zu sagen hat und dass er sein Quantenhandy einschalten soll. Außerdem machen sie sich die Basen aus, in denen sie messen werden. Sie entscheiden sich für $+$ und \times . Alice erzeugt jetzt polarisierte Photonen und wählt dafür zufällig mit gleicher Wahrscheinlichkeit eine der vier möglichen Polarisationen. Sie notiert sich ihre Messung, also die Polarisation und die verwendete Basis. Die Photonen werden dann zu Bob weitergeleitet. Dieser misst ebenfalls in Zufälliger Reihenfolge entweder in der Basis $+$ oder \times und notiert sich

Polarisation und Basis. Nachdem sie genügend Photonen übertragen haben, machen sie sich für jede Polarisation einen Wert aus. In diesem Beispiel heißt \leftrightarrow und \nearrow gleich 0 und \updownarrow und \swarrow gleich 1. Jetzt vergleichen Alice und Bob ihre Basen und verwenden nur diejenigen Messungen, bei denen sie die gleiche Basis verwendeten. Die übrigen werden verworfen. Sie werden also ca. 50% ihrer Messergebnisse verwenden können. Ein Beispiel ist in Tabelle 10 zu sehen. Die Spalte „Alice“ zeigt an, was Alice gemessen hat, die Spalte „Bobs Basis“, welche Basis Bob verwendete, „Bob“ was Bob gemessen hat, „Gleich?“ zeigt an, ob es die selbe Basis war und „Bit“ welches Bit dem schließlich zugeordnet wird.

Tabelle 10: Eine Beispielmessreihe im BB84-Protokoll

Alice	Bobs Basis	Bob	Gleich?	Bit
\swarrow	\times	\swarrow	ja	0
\updownarrow	\times	\swarrow	nein	-
\swarrow	$+$	\updownarrow	nein	-
\leftrightarrow	$+$	\leftrightarrow	ja	0
\swarrow	\times	\swarrow	ja	0
\updownarrow	$+$	\updownarrow	ja	1
\updownarrow	\times	\swarrow	nein	-
\swarrow	$+$	\leftrightarrow	nein	-
\swarrow	\times	\swarrow	ja	1
\leftrightarrow	$+$	\leftrightarrow	ja	0
\vdots	\vdots	\vdots	\vdots	\vdots

Nachdem sie genügend Bits übertragen haben, vergleichen sie über einen potentiell unsicheren Kanal ihre verwendeten Basen. Bits bei denen sie unterschiedliche Basen verwendeten, werden verworfen (in der Tabelle als - gekennzeichnet). Die übrigen Bits verwenden sie aber nicht gleich für den Key. Zunächst müssen sie herausfinden, ob sie belauscht wurden, oder nicht. Was wäre, wenn sie belauscht wurden? Alice polarisiert ein Photon zufällig in $+$. Eve hat sich zwischen Alice und Bob gesetzt und möchte den Schlüssel herausfinden. Zu diesem Zeitpunkt, weiß aber Eve nicht, welche Basis Alice verwendet hat und muss sich zufällig entscheiden. Wählt sie zufällig die richtige Basis, erhält sie das gleiche Ergebnis wie Alice. Wählt sie die falsche, bekommt sie ein willkürlich anderes und das wird in 50% der Fälle passieren. Wählt Alice die Basis $+$ und misst \updownarrow und Eve die Basis \times und misst z.B. \swarrow und schickt dann an Bob ein \swarrow -Photon und der misst aber ebenfalls in $+$ und erhält nicht \updownarrow sondern \leftrightarrow , was wiederum in 50% der Fälle passiert, in denen Eve die falsche Basis verwendet hat, dann merken Alice und Bob, dass etwas faul ist, denn wenn sowohl Alice, als auch Bob

+ verwendeten, aber unterschiedliche Ergebnisse erhalten, dann war das entweder ein Messfehler, oder das Photon hat auf der Reise zwischen Alice und Bob „vergessen“, welche Polarisation es bezüglich der HV-Basis hat. Da Alice und Bob wissen, wie gut ihre Quantenhandys sind, werden sie bei zu großen Fehlerraten annehmen, dass sie belauscht wurden und die Bits verwerfen. Eve hat somit keine Chance, unbemerkt den Schlüssel mitzulauschen und somit den Inhalt der Kommunikation zu erfahren. Alles was sie machen kann, ist die Kommunikation zu stören und zu verhindern. [7]

5 Unterricht

In den einzelnen Kapitel zu den verschiedenen Verschlüsselungsalgorithmen, habe ich versucht, Unterrichtssequenzen zu planen, die direkt das Verständnis und die Anwendung der jeweiligen Methode verdeutlichen sollen. An dieser Stelle möchte ich Unterrichtssequenzen beschreiben, die nicht so speziell auf den einen oder anderen Algorithmus eingehen, sondern einen größeren und gröberen Überblick über kryptographische Software bieten sollen. Außerdem soll ein Grundverständnis und eine gewisse grundlegende Vernunft vermittelt bzw. geweckt werden, um potentielle Gefahren in Bezug auf Kommunikation, speziell in Computernetzwerken, einschätzen zu können.

Nachdem in einer Einführung erklärt, erarbeitet oder auf sonstige Art vermittelt wurde, um was es bei Kryptographie prinzipiell geht, sollen die Schülerinnen und Schüler sich selbstständig überlegen, wo im Alltag man auf sie stoßen kann, oder wo die Schüler und Schülerinnen gerne hätten, dass verschlüsselt wird.

5.1 Banken

Ein Bereich des täglichen Lebens, an den bei diesem Kapitel meist als erstes gedacht wird, ist das Bankwesen. Zunehmend laufen Transaktionen nicht mehr zwischen Kunden und Bankangestellten ab, sondern zwischen Kunden vor dem eigenen Computer oder vor einem Computer der Bank (Bankomat) und *der Bank*, wobei *die Bank* eine Blackbox ist, von der man eigentlich nicht weiß, wie sie funktioniert. Was aber jeder weiß ist, dass man einen vierstelligen PIN eingeben muss. Angenommen die Übertragung des PINs ist sicher, was muss noch beachtet werden? Reichen vier Stellen aus? Was für Angriffe auf den PIN und die Karte können passieren? Und wie wehrt man sich effektiv dagegen? Logische Überlegungen und Recherche werden den Schülerinnen

und Schülern hier viele Strategien liefern, über die einige Referate gehalten werden können.

Die Frage, ob ein vierstelliger PIN ausreichend ist, kann zum Beispiel erst dann gut beantwortet werden, wenn man weiß, wie viele Chancen man hat ihn einzugeben. Zur Zeit ist es so, dass man mit der Karte eine Verbindung zu einem Bankserver aufbaut, auf dem man sich dann mit dem PIN identifiziert. Das heißt, nur der Besitz der Karte, ob im Original oder in Form einer Kopie, bringt einem Angreifer nichts. Er muss auch den PIN wissen. Nach drei Fehlversuchen, wird die Karte gesperrt und sofern sie an einem legitimen Automaten verwendet wurde, auch eingezogen. Bei einem vierstelligen PIN gibt es 10000 mögliche Zahlenkombinationen. Von 0000 bis 9999. Die Chance, den PIN bei drei Versuchen zu erraten liegt somit bei $(1/10000) + (1/9999) + (1/9998)$ bzw. bei 0,03 Prozent. Eine Strategie, die Chancen für einen unehrlichen „Finder“ der Karte noch weiter zu senken, wäre zum Beispiel, einen PIN, aber natürlich nicht den eigenen, auf die Karte zu schreiben. Der Finder denkt sich „Jackpot, der war wirklich so dumm, seinen PIN auf die Karte zu schreiben!“ und probiert sie gleich aus. Natürlich ist der PIN falsch. Denkt er sich von Anfang nicht, dass das der richtige PIN ist, wurde die Sicherheit auf $(1/9999) + (1/9998) + (1/9997)$ gesenkt, was noch immer rund 0,03 Prozent sind. Die meisten Gelegenheitsnutzer werden die Zahl auf der Karte aber für den PIN halten. Nach dem ersten Fehlversuch werden wahrscheinlich die meisten Leute zunächst an sich selbst zweifeln, sich denken, dass sie sich vertippt haben und nochmal den selben, falschen PIN eingeben. Wenn sie jetzt mitkriegen, dass der PIN falsch ist, haben sie nur noch einen Versuch und können nur eine Zahl ausschließen, was ihnen eine Chance von $(1/9999)$ oder 0,01 Prozent einräumt, also nur mehr ein Drittel von vorhin. Wenn der falsche PIN dann noch dazu ausschließlich aus Zahlen besteht, die auf dem Kopf stehend auch Sinn ergeben, denkt sich der Finder eventuell nach dem zweiten Versuch, dass er die Karte verkehrt herum hielt und vergibt seinen letzten Versuch mit der verdrehten Kombination.

5.2 WLAN

Sich darauf zu verlassen, dass physische Kabel abhörsicher sind, weil es aufwendig ist sie anzuzapfen ist zwar naiv, reicht aber für die Bedürfnisse der meisten Nicht-Agenten aus. Bei kabelloser Übertragung von Daten werden aber dennoch die meisten Menschen, die normalerweise mit Computersicherheit nicht so viel zu tun haben, ein wenig misstrauisch. Mittlerweile ist es so, dass man zu jedem Vertragsschluss mit einem ISP auch zusätzlich zu dem DSL oder Kabel-Modem auch einen WLAN-Router

erhält. Die Übertragung zwischen Router und Laptop oder PC kann man dann verschlüsseln, damit sich der Nachbar oder sog. „War Driver“ nicht ins heimische Netz einschleichen können und auf Kosten des Anschlussinhabers ins Internet geht. Dabei geht es nicht unbedingt nur um finanzielle Kosten, sondern auch um rechtliche Gefahren. Aus Deutschland gibt es schon Fälle, in denen Rentner wegen Urheberrechtsverletzung verklagt und verurteilt wurden, weil sich unbekannte Dritte über ihren Anschluss per WLAN Lieder des „Gangsta-Rappers“ Bushido heruntergeladen hatten¹⁸. Es ist dabei kaum einer Erwähnung wert, dass das besagte Rentnerhepaar gar nicht wusste, wer oder was Bushido ist.

Eine mit WEP gesicherte WLAN-Verbindung ist so gut wie nicht gesichert. Es ist nur eine Frage von Minuten, bis der Angreifer den WEP-Schlüssel hat und sich ins Netzwerk einloggen kann. Das einzige, was WEP einem bietet ist eine gewisse rechtliche Sicherheit, da ein erwischter Angreifer nicht behaupten kann, er habe sich unabsichtlich zum Netzwerk des Users Zugriff verschafft, was bei offenen und ungesicherten WLAN- Routern durchaus passieren kann, wenn Laptops oder Smartphones so konfiguriert sind, sich automatisch mit verfügbaren WLAN-Netzen zu verbinden.

Auch der Nachfolger von WEP, WPA ist nicht mehr aktuellster Stand der Technik und kann neuesten Forschungsergebnissen nach in günstigen Fällen schon in wenigen Minuten geknackt werden. Man sollte daher WPA2 verwenden und bei einem pre-shared key einen ausreichend guten Schlüssel verwenden.

5.3 Passwörter

Ein wichtiges Kapitel im Unterricht, wenn es um Computersicherheit geht, sind Passwörter. Alle Kosten und Mühen, die in die Entwicklung von sicheren Algorithmen und fehlerarmer Software fließen, können hier zunichte gemacht werden. Seine Festplatte mit AES mit 128 Bits Key zu verschlüsseln bringt gar nichts, wenn die verwendete Passphrase abc123 ist. Die Passphrase ist sozusagen die Schnittstelle zwischen Mensch und den verschlüsselten Daten. Und hier liegt auch das Problem: Mensch. Wir sind nicht unbedingt gut dafür ausgerüstet, sichere Passphrases zu erzeugen und sie uns zu merken. Denn sichere Passphrases sind lang und bestehen aus Zeichen eines möglichst großen Alphabets die zufällig angeordnet sind. Lang bedeutet, mindestens 15, besser aber 20 Zeichen, großes Alphabet heißt alle Klein- und Großbuchstaben des englischen Alphabets, alle Ziffern und möglichst viele Sonderzeichen. Hier wurde jetzt

¹⁸<http://tinyurl.com/yfhxbzg>

somit das Ideal eines Passwortes skizziert. Als Lehrer muss einem klar sein, dass nur sehr wenige Schüler und Schülerinnen versuchen werden, diesem zu entsprechen. Deshalb ist es mit Sicherheit nicht verkehrt, die absoluten Todsünden was Passphrases angeht aufzuzählen, damit wenigstens sie vermieden werden.

Gute und schlechte Passwörter

Wer seine Festplatte verschlüsselt, auf seinem privaten Rechner SSH offen lässt oder ähnliches, weiß, was er oder sie tut, oder *sollte* es wissen. Hier werden keine Ausreden toleriert, 15 bis 20 Zeichen, zufällig gewählt.

Alle anderen, die Passwörter für ihren E-Mail-Account, Facebook, und der gleichen verwenden: 8 bis 10 Zeichen dürfen es schon sein. Man sollte zumindest versuchen, das ein oder andere Sonderzeichen einzubauen, ein bis zwei Ziffern und Groß- und Kleinbuchstaben. `F1ov0ut?t9` ist ganz gut. Um sich das zu merken, kann man den Ziffern Klänge zuordnen. Ein Einser sieht aus wie ein kleines L. Dieses Passwort würde ich mir als den Satz „Flov Out? t9“ merken. Wobei Flov natürlich keine Bedeutung hat, aber das haben Namen auch nicht, trotzdem merkt man sie sich. Und t9 ist das Wörterbuch das man für Textnachrichten auf einem Handy verwenden kann. Mit solchen Eselsbrücken kann man sich auch relativ komplexe Passwörter ganz gut merken.

Passwörter, die man vermeiden sollte, sind prinzipiell alle Wörter, die in Wörterbüchern zu finden sind und alle Wörter, die aus Wörtern zusammengesetzt sind, die in Wörterbüchern zu finden sind. `Spiegel` ist zum Beispiel schlecht, weil dieses Wort in Wörterbüchern zu finden ist. `Fragenspiegel` ist zwar kein sinnvolles Wort, aber aus zwei sinnvollen Wörtern zusammengesetzt und damit unbrauchbar.

Auch sehr häufige Passwörter, wie `abc123` und `password1`, sind zu vermeiden. Ebenso alle Passwörter, die einem Muster auf einer Tastatur entsprechen. Zum Beispiel `asdfjklö`, oder `2wsx3edc`. Passwörter die aus Informationen über einen selbst bestehen, sind natürlich auch Tabu, entweder, weil sie gegen die Wörterbuch-Regel verstoßen, was bei den meisten Vornamen der Fall ist, oder weil die Informationen für sehr viele Menschen einsehbar war und somit leicht erratbar ist. Geburtsdatum, Name des Haustiers, Wohnort, Lieblingsfilm usw. sollten nicht verwendet werden.

Umgang mit Passwörtern

Passwörter aufzuschreiben ist mit Vorsicht zu genießen. Prinzipiell ist das keine gute Idee. Einen Zettel kann man leicht verlieren, die Stasi kann einem die Wohnung aus-

räumen und ihn mitnehmen, usw. Besser ist es, das Passwort nicht aufzuschreiben. Wenn man es trotzdem tut, ist es klar, dass man ein paar Regeln beachten muss. Erstens, vor fremden Blicken schützen. Nicht auf einen Zettel schreiben und im Arbeitszimmer liegen lassen. Wenn man auf einen Zettel schreibt, darauf achten, dass man keine Tinten- oder Durchdrucksspuren hinterlässt. Den Zettel am besten immer bei sich haben, oder an einem *wirklich* sicheren Ort verstecken. Auch sinnvoll ist, dass man den Zettel so präpariert, dass man sofort merkt, wenn jemand das Passwort gelesen hat. Zum Beispiel versiegeln. Ebenfalls keine schlechte Idee ist, ständig die Mittel bei sich zu haben, um den Zettel vollständig zu vernichten. Schlucken ist keine gute Idee, Papier ist sehr widerstandsfähig. Verbrennen ist besser, ist aber nicht unter allen Umständen rasch machbar. Was man vermeiden sollte ist, Passwörter auszudrucken. Viele Drucker haben einen Zwischenspeicher, der oft auch persistent ist. Deshalb (nehme ich an) werden bei Hausdurchsuchungen immer auch die Drucker mitgenommen, obwohl man sich normalerweise denken würde, dass das sinnlos ist.

Wer Passwörter computertechnisch verwalten will, muss auch einige Sachen beachten. Auf keinen Fall das Passwort im Klartext auf die Festplatte schreiben. Dies birgt nämlich gleich mehrere Gefahren. Erstens, der Computer ist wahrscheinlich auf kurz oder lang mit dem Internet verbunden, was ihn prinzipiell angreifbar macht. Und selbst wenn nicht, ist es relativ schwer, Informationen von einer Festplatte wieder zu löschen, vor allem bei Journaling-Dateisystemen wie ext3 oder reiserfs. Am besten man speichert auf einem in FAT formatierten USB-Stick und *nicht* im Klartext, sondern verschlüsselt. Hierfür gibt es entweder Tools wie `aespipe`, oder extra dafür geschriebene Software, wie das für praktisch jede Plattform verfügbare KeePass¹⁹. Diese Software bietet ein übersichtliches Interface zum Speichern von Passwörtern, gibt Auskunft über den Sicherheitsgrad, lässt einen Ablaufdaten definieren und hat noch viele weitere Funktionen, die man eventuell brauchen kann. Die wichtigste ist aber, dass mit AES oder Twofish verschlüsselt wird.

Die meisten Menschen haben über die ganze Welt verstreut, unzählige Logins. E-Mail-Accounts, Social Networks, Arbeitsplatzrechner, Internetforen, YouTube und so weiter. Die Liste ist beliebig lang fortsetzbar. Das Problem mit guten Passwörtern ist, dass man sie sich nur recht schwer merkt. Deshalb tendieren Menschen dazu, für jeden Login das selbe Passwort zu verwenden. Strategisch ist das nicht besonders gut, da so ein Angreifer relativ einfach, viele der Logins übernehmen kann, wenn er nur einen knackt. Wenn zum Beispiel bei Facebook eingebrochen wird und für den

¹⁹<http://keepass.info/>

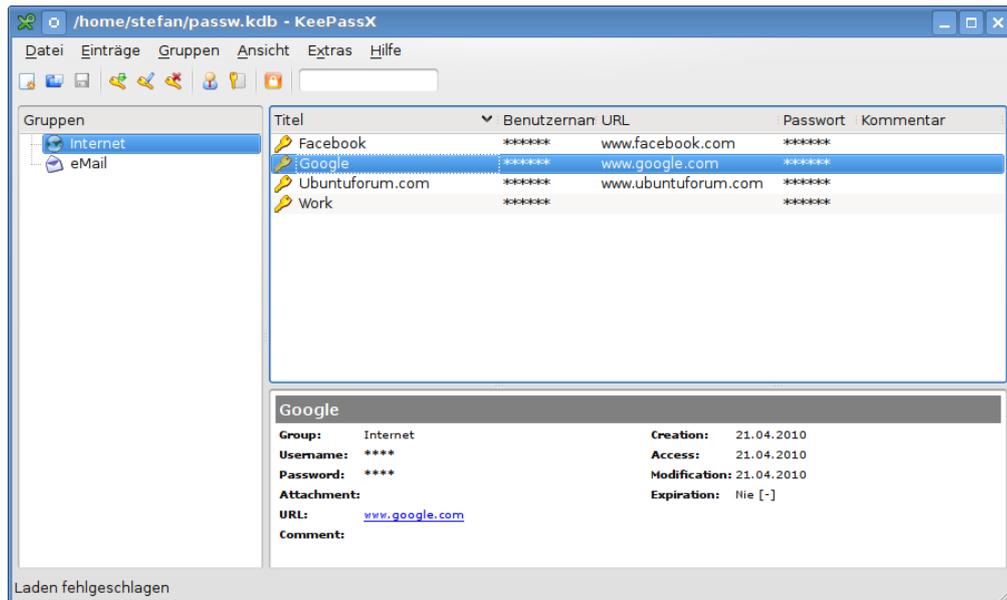


Abbildung 18: Screenshot von KeePass

Facebook-Account und die dafür verwendete E-Mail-Adresse das selbe Passwort verwendet wurde, sind mit einem Schlag gleich zwei Logins weg. Mit der E-Mail-Adresse kann sich der Angreifer dann neue Passwörter in diversen Foren schicken lassen und den eigentlichen Account-Inhaber aussperren. Ein Kompromiss zwischen, für jeden Account das selbe Passwort verwenden und sich für jeden Account ein eigenes, gutes Passwort merken zu müssen ist, die Internetadresse des Services in das Passwort einzubauen und sich dann nur die Schritte zu merken, wie die Internetadresse eingebaut wird. Wenn man das Passwort von vorhin zum Beispiel bei Facebook verwenden will, dann könnte man folgendes machen: Den ersten Buchstaben in eine Zahl umwandeln (A wird zu 1, B zu 2, ... F wird zu 6) und vor das Passwort schreiben. Den letzten Buchstaben in eine Zahl umwandeln, Quersumme des eigenen Geburtsdatums dazurechnen, die Quersumme des Handy-PINs abziehen, Modulo 26 nehmen, zurück in Buchstaben umwandeln und nach einem Unterstrich an das Passwort hängen. Das Facebook-Passwort für jemanden, der am 07. Mai 1985 Geburtstag hat und dessen Handy-PIN 9031 ist, wäre nach diesem Verfahren `6F1ov0ut?t9_v`. Dieses Verfahren zielt natürlich darauf ab das Verfahren, nachdem man die Internetadresse einbaut geheim zu halten und verstößt somit gegen das Kerckhoffs'sche Prinzip (zumindest der Handy-PIN streut noch ein wenig Kerckhoff ein), es soll uns aber auch nur einen kleinen Zeitpuffer schenken, den wir dazu nutzen können um bei den anderen Accounts neue Passwörter zu setzen. Diese und ähnliche Tipps und Tricks findet man Zu-

hauf, wenn man eine Internetsuche mit Schlagwörtern wie „Password cracking“, oder „Password strength“, absetzt. Obiges Verfahren habe ich mir selbst ausgedacht, der Mechanismus, Servicenamen oder URLs in Passwörter einzubauen, ist aber gängige Praxis und nicht von mir erfunden.

Erzeugung von Passwörtern

Software Eine gute Möglichkeit, sichere Passwörter zu generieren, sind Passwortgeneratoren. Für das Passwort aus dem vorherigen Beispiel habe ich `apg`²⁰ verwendet. Das Schöne an `apg` ist, dass man damit aussprechbare Passwörter erzeugen kann, die trotzdem noch sicher sind. Eine Beispielausgabe von `apg` ist in Listing 23 zu sehen.

Listing 23: Passwort generieren mit `apg`

```
$ apg

Please enter some random data (only first 16 are significant)
(eg. your old password):>
debyivAm7 (deb-yiv-Am-SEVEN)
Ci0f40cmy (Ci-Of-FOUR-0c-my)
Dydturpoing3 (Dyd-turp-oing-THREE)
faibDabcov6 (faib-Dab-cov-SIX)
yu2Heveek5 (yu-TWO-Hev-eeek-FIVE)
Phodju0bjoz5 (Phod-ju-0b-joz-FIVE)
```

Ohne weitere Optionen werden sechs aussprechbare, mittellange Passwörter generiert, nachdem man aufgefordert wurde, zufällige Tasten anzuschlagen, um genug Entropie für die Generierung zu haben. Gleichzeitig werden Vorschläge gemacht, wie man sich das Passwort merken kann. Mit dem Schalter `-a 1` wird kein Wert mehr auf Aussprechbarkeit gelegt. Mit `-m` und `-x` kann die minimale bzw. maximale Länge festgelegt werden. `-n` gibt an, wie viele Passwörter man generieren möchte. Der Schalter `-M` gibt den Modus an, das heißt, man kann definieren, ob Sonderzeichen, Nummern, Großbuchstaben (`capital letters`) und Kleinbuchstaben (`lower letters`) verwendet werden *müssen* (Großbuchstabe) oder *sollen* (Kleinbuchstabe). Um ein Passwort mit 15 bis 20 Zeichen zu generieren, in dem Groß- und Kleinbuchstaben, Ziffern und Sonderzeichen vorkommen *müssen*, gibt man ein:

²⁰<http://www.adel.nursat.kz/apg/>

Listing 25: Passwortgenerator in Bash

```
#!/bin/bash

if [ -z $1 ]; then
    length=15
else
    length=$1
fi

symbols=( 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 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 0 1 2 3 4 5 6 7 8 9 { } \ ( \ ) ! \ " \ # \ $ \ % \ & \ ' \
+ , - . / : \ ; \ < \ = \ > ? @ \ [ \ \ \ ] ^ _ \ ` \ | \ ~ \ )

for i in `seq 1 $length`; do

    echo -n ${symbols[$(( $RANDOM % 94 )) ]}

done

echo ""
```

Listing 24: Passwort generieren mit apg

```
$ apg -a 1 -n 1 -m 15 -x 20 -M SNCL
4$~+K<?5t*04p821sa{E
```

Wenn man im Unterricht beim Thema Programmierung ist, kann man auch einen Passwortgenerator in einer beliebigen Sprache schreiben lassen. In Listing 25 ist eine in unter 15 Minuten geschriebene Variante in Bash.

Münzen Wer kennt es nicht? Man sitzt mit einem Freund in einer Bar und plötzlich hat man das Bedürfnis nach einem starken Passwort. Leider hat man nichts bei sich als seine Brieftasche, Handy und Schlüssel. Wer in Europa lebt hat Glück, denn unser Kleingeld eignet sich hervorragend, um Schlüssel zu erzeugen. Der Euro hat genau acht Münzen, nämlich 0,01, 0,02, 0,05, 0,1, 0,2, 0,5, 1 und 2 Euro. Auf eine kann man verzichten, man braucht für das Folgende nur sieben Münzen. Hier wird die 2 Euro Münze weggelassen. Was man jetzt machen kann ist, jede Münze einer Stelle einer 7 Bit-Zahl zuzuordnen. 1 Cent wird der ersten Stelle, 2 Cent der zweiten usw. zugeordnet. Jetzt nimmt man alle Münzen in die Hände, schüttelt gut durch und wirft sie kontrolliert auf den Tisch. Was man jetzt erhalten hat, ist eine 7 Bit Zufallszahl.

Kopf wird dabei als 0 interpretiert und Zahl als 1. Zum Beispiel das Ergebnis Z, K, K, K, K, Z, K (Z=Zahl, K=Kopf), wobei von links nach rechts gelesen die Münzen in absteigender Reihenfolge angeordnet sind, wäre die Binärzahl 1000010, was Dezimal $0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 = 66$ entspricht, bzw. in den ASCII-Code interpretiert B. Das Ganze macht man nun 15 bis 20 Mal und hat so ganz einfach ein wirklich gutes Passwort für spätere Kommunikation erzeugt. Für alle Zahlen die kleiner als 33 sind und die Zahl 127 muss man nochmal werfen, denn hier sind in der ASCII-Tabelle Steuerungszeichen definiert.

Sätze Ein weiterer, recht beliebter Trick, um einigermaßen sichere und sich merkbare Passwörter zu generieren, ist einen Satz zu erzeugen und jeweils die Anfangsbuchstaben der Wörter als Zeichen für das Passwort zu verwenden. Sehr bekannte Zitate sollte man nicht nehmen, am besten ist hier Nonsense. Zum Beispiel der Satz „Wenn ich auf dem Boot zum Kopierer ein Schnitzel finde, tauche ich einen Almdudler.“ ergibt keinen Sinn, ist also relativ schwer zu erraten. Das Passwort, das sich aus ihm ergibt ist `Wia dBzK e Sf, tieA..`. Es hat Groß- und Kleinbuchstaben und Satzzeichen. Jetzt kann man es noch mit Zahlen und Sonderzeichen garnieren. Zum Beispiel statt B einen Achter verwenden und das e einklammern. Somit hat man `Wia d8zK(e)Sf, tieA..`. Achtzehn Zeichen, Groß- und Kleinbuchstaben, Sonderzeichen und Zahlen, wirklich nicht schlecht. Aber trotzdem nicht zufällig und anfälliger als zufällige Passwörter. Das ist wieder so ein Fall von „es ist schwer, zu erkennen, was schwer ist“ (frei nach Bart Preneel²¹). Für jemanden, der den Ursprung der Passphrase nicht kennt, sieht sie sehr zufällig aus. Tatsächlich ist es aber so, dass die Reihenfolge von Groß- und Kleinbuchstaben nicht zufällig ist. Selbst wenn der Satz *semantisch*, also inhaltlich keinen Sinn ergibt, so ist er doch *syntaktisch* korrekt und lässt auf die Position von Verben, Adverbien, Hauptwörtern usw. schließen, was dann die Position der Groß- und Kleinbuchstaben verrät. Wenn jemand die Passphrase angreift, nimmt er zunächst an, der Benutzer ist naiv und probiert alle Wörter in Wörterbüchern durch. „Wörterbücher“ ist aber nicht im engeren Sinn wie Duden zu verstehen, sondern es handelt sich um Dateien mit allen Wörtern, die für Passwörter verwendet werden können, also auch so etwas wie abc123. Hat man so ein Passwort gewählt, ist das in wenigen Stunden mit einer Wörterbuchattacke geknackt. Wenn nicht, kommt der nächste Schritt. Der Angreifer unterstellt ein wenig mehr Vorsicht und probiert das Sätze-Schema mit berühmten Zitaten, den Lieblingsbüchern des Opfers (die er von

²¹<http://homes.esat.kuleuven.be/~preneel/>

Facebook kennt oder weil er in die Wohnung eingebrochen ist und das Bücherregal fotografiert hat) usw. Ist die Passphrase noch immer nicht geknackt, kommen solche Tricks dazu, wie oben beschrieben: Buchstaben durch Zahlen ersetzen, Sonderzeichen einfügen etc. Je mehr von den Tricks man angewendet hat, umso länger dauert das Erraten der Passphrase.

Trotzdem sind solche Angriffe wesentlich effizienter als Brute-Force-Attacken, also das stupide Durchprobieren *aller* Möglichkeiten. Ist das Passwort wirklich zufällig, (und zufälliger Weise auch gut) dann bleibt nur Brute-Force übrig und bei entsprechender Länge dauert das eben Jahrzehnte, Jahrhunderte oder länger.

Herauszufinden, wie stark ein Passwort ist, ist nicht immer leicht, vor allem dann nicht, wenn die Zufälligkeit, mit der die Zeichen für das Passwort ausgewählt wurden, nicht genau bestimmbar ist. Am zuverlässigsten was das angeht, sind gute Würfe mit einer Münze, bei der sowohl Kopf, als auch Zahl, gleich wahrscheinlich geworfen werden. Der Rand der Münze ist abgerundet, daher kann sie nicht hochkant stehen. Außerdem wird davon ausgegangen, dass die Münze von einem Roboter geworfen wird, um den „Bingo-Effekt“, der weiter oben beschrieben wurde, zu vermeiden. Wenn man davon ausgeht, dass die Zeichen in einem Passwort zufällig gewählt wurden, dann kann man die Stärke des Passworts dadurch ausdrücken, wie oft man eine Münze werfen müsste, um die gleiche Sicherheit zu erreichen. Mit anderen Worten, wie hoch ist die Bit-Stärke des Passworts.

Für ein Passwort, das aus n Zeichen besteht, die aus einem Alphabet ausgewählt wurden, in dem X Symbole sind, berechnet man die Stärke mit der Formel²²

$$H = n \log_2(X) = n \frac{\log(X)}{\log(2)}. \quad (35)$$

Möchte man zum Beispiel eine Stärke von 128 Bits erreichen und verwendet dafür alle 95 druckbaren ASCII-Zeichen (siehe Listing 25 plus Stern, der weggelassen wurde), dann setzt man in die Formel ein:

$$128 = n \log_2(95) \quad (36)$$

$$n = 128 / \log_2(95) \quad (37)$$

$$n = 19,48... = 20 \quad (38)$$

²²http://en.wikipedia.org/w/index.php?title=Random_password_generator&oldid=362710975

Man braucht somit 20 zufällige Zeichen, die alle aus der Menge der druckbaren ASCII-Zeichen ausgewählt werden müssen, um ein Passwort zu erreichen, das einem 128 Bits Key entspricht.

Würfel Um mit einem Würfel sichere, relativ leicht merkbare Passphrasen zu erstellen, wurde Diceware²³ entwickelt. Das Prinzip ist recht einfach: Man würfelt fünf mal und erhält dadurch eine fünfstellige, senäre Zahl. Diese Zahl schlägt man anschließend in einer Wortliste, die man aus dem Internet beziehen kann, nach und erhält so ein Wort. Dies wiederholt man noch mindestens weitere vier Mal und generiert sich somit eine Passphrase, die aus fünf Wörtern besteht. Da man sehr genau weiß, wie viel Entropie jedes Wort der Passphrase hinzufügt, kann man so recht einfach bestimmen, wie sicher die Passphrase ist. Da jedes Wort 12,9 Bit Entropie hinzufügt, benötigt man um mindestens 64 Bit zu erhalten, mindestens fünf Worte. Wenn man durch Zufall viele sehr kurze Worte erwürfelt hat, oder die fünf Worte einen korrekten Satz bilden, sollte man noch einmal von vorne anfangen.

Ein Beispiel: Zunächst habe ich die fünfstellige Zahl 14352 gewürfelt. In der Datei `diceware_german.txt`²⁴ nachgeschaut, erhält man das Wort `baten`. Dann ging es weiter mit...

11231 48
66422 zj
55255 skobje
23634 fass

Zusammen hat man somit die Passphrase `baten48zjkskobjefass`. Laut Diceware-Homepage²⁵ sind 14 Stellen, mit Leerzeichen zwischen den Wörtern, genug, daher ist die obige Passphrase einsatzbereit und braucht nicht noch ein Wort.

²³<http://de.wikipedia.org/w/index.php?title=Diceware&oldid=66338334>

²⁴http://world.std.com/~reinhold/diceware_german.txt

²⁵<http://world.std.com/~reinhold/diceware.html>

Literatur

- [1] B. Jack Copeland. *The Essential Turing*. Oxford University Press, 1 edition, 2004.
- [2] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 1 edition, 2002.
- [3] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 1 edition, 2004.
- [4] Heise.de. Alcatel-Lucent vermisst Datenträger mit sensiblen Mitarbeiterinformationen, 2007. (Online, Stand 6. Juli 2009).
- [5] David A. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Simon & Schuster, 1 edition, 1997.
- [6] Richard A. Mollin. *An Introduction to Cryptography*. Chapman & Hall/CRC, 2 edition, 2007.
- [7] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 1 edition, 2000.
- [8] NIST. Data Encryption Standard. Technical report, National Institute of Standards and Technology, 1999.
- [9] NIST. Advanced Encryption Standard. Technical report, National Institute of Standards and Technology, 2001.
- [10] NIST. Digital Signature Standard. Technical report, National Institute of Standards and Technology, 2009.
- [11] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Technical report, Massachusetts Institute of Technology, 1977.
- [12] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc., 2 edition, 1996.
- [13] Bruce Schneier. The Eternal Value of Privacy. *Wired.com*, 18. Mai 2006.

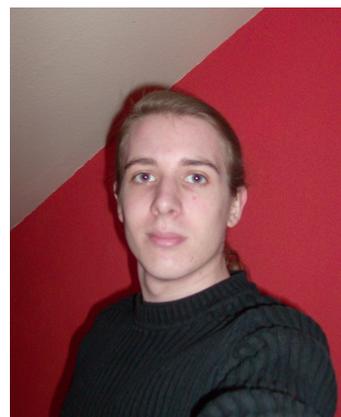
- [14] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. Technical report, 1998.
- [15] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor Books, 1 edition, 2000.
- [16] Richard Stallman. The free software definition, 2009. (Online, Stand 21. Juli 2009).
- [17] Neal Stephenson. *Cryptonomicon*. Harper Perennial, 1 edition, 1999.
- [18] Paul A. Tipler and Gene Mosca. *Physik für Wissenschaftler und Ingenieure*. Spektrum Akademischer Verlag, 2 edition, 2006.
- [19] Wikipedia. Geschichte der Kryptographie, 2009. (Online, Stand 20. Mai 2009).

Lebenslauf

Stand: 25. Mai 2010

Angaben zur Person

Nachname, Vorname GRUBER, Stefan
Adresse Kl. Neusiedlerstraße 4/12
2401 Fischamend
Telefon 0650-9016144
E-Mail stef.gruber@gmail.com
Staatsangehörigkeit Österreich
Geburtsdatum 12. Juni 1986



Schul- und Berufsbildung

Studium Universität Wien, TU Wien, 2005 bis heute
Fächer UF Physik und UF Informatik
Zivildienst Rettungs- und Krankentransport-Sanitäter
Dienststelle Rotes Kreuz Schwechat, 2004 bis 2005
Gymnasium BRG Schwechat, 1996 bis 2004
Volksschule VS Münnichplatz, Wien, 1992 bis 1996

Berufserfahrung und persönliche Fähigkeiten

- Während des Zivildiensts Rettungs- und Krankentransport.
- Während des Studiums PC-Raum-Betreuer beim Zentralen Informatikdienst der Universität Wien.