# RSA & Co. in der Schule

Moderne Kryptologie, alte Mathematik, raffinierte Protokolle

Neue Folge – Teil 7: Alternativen zu RSA oder: Diskreter Logarithmus statt Faktorisierung

von Helmut Witten, Ralph-Hardo Schulz und Bernhard Esslinger

**In den vorangehenden Folgen dieser Beitragsserie haben wir uns mit unterschiedlichen Aspekten bei der Behandlung von RSA im Schulunterricht beschäftigt, u.a.: Wie funktioniert der RSA-Algorithmus? Warum funktioniert RSA? Wie sicher ist RSA?**

**Dort haben wir jeweils auch kleine Ausflüge in die algorithmische Zahlentheorie unternommen, einem faszinierenden Wissenschaftsgebiet an der Grenze zwischen Mathematik und theoretischer Informatik. Die Wurzeln der Zahlentheorie reichen bis ins Altertum wie beispielsweise beim euklidischen Algorithmus, dessen Verfahren von Euklid in seinem Werk *Die Elemente* bereits im dritten Jahrhundert v. Chr. beschrieben wurde. Aufgrund der modernen Kryptologie hat die Zahlentheorie enorme praktische Bedeutung erlangt.**

## Jenseits von RSA

RSA ist das wichtigste und am weitesten verbreitete asymmetrische Krypto-System, aber keineswegs das einzige. Deswegen wollen wir hier auf die Alternativen schauen.

Dies führt uns zurück zu den Anfängen der asymmetrischen Kryptografie, die mit den drei Namen Diffie, Hellman und Merkle verbunden sind (siehe Bild 1). Gleichzeitig wird auch die politische Dimension dieser Wissenschaft deutlich: Diffie und Hellman ging es darum, die Kryptografie der Öffentlichkeit zugänglich zu machen. Ihre Arbeit markiert daher auch den Anfang der nicht von Geheimdiensten kontrollierten Entwicklung kryptografischer Verfahren. Der Artikel von Whitfield Diffie *The First Ten Years of Public-Key Cryptography* ist eine Beschreibung der Anfänge aus erster Hand (vgl. Diffie, 1988). Eine detaillierte Vorstellung findet sich ebenfalls in dem lesenswerten Buch von Simon Singh, dem bekannten Autor populärwissenschaftlicher Werke (vgl. Singh, [10]2011). Eine ausführlichere Darstellung der politischen Aspekte der Public-Key-Kryptografie kann man in dem Buch *Crypto* von Steven Levy nachlesen (vgl. Levy, 2001).

Martin Hellman (geb. 1945 in New York) ist uns bereits in Folge 3 *RSA und die elementare Zahlentheorie* begegnet (vgl. Witten/Schulz, 2008, S.69). Er war bis zu seiner Emeritierung 1996 Professor für Elektrotechnik an der Stanford-Universität mit Interesse an Fragen der Kryptologie und setzt sich seit vielen Jahren für Bürgerrechte in Zeiten der globalen Vernetzung ein – eine hochaktuelle Frage!

Whitfield („Whit“) Diffie wurde 1944 in Washington, D.C., geboren. Bereits im Alter von zehn Jahren ließ er sich von seinem Vater, einem Professor am New York City College, alle Bücher zu Fragen der Kryptologie mitbringen, um sich Einblicke in diese faszinierende Wissenschaft zu verschaffen (vgl. Wikipedia – Stichwort „Whitfield Diffie“). Während seines Mathematik-Studiums am MIT beschäftigte er sich auch mit Künstlicher Intelligenz und kam in Kontakt mit der dort aktiven Hacker-Szene. Ab 1972 verlagerten sich seine Interessen endgültig zu Kryptologie und Datenschutz (engl.: *privacy*). Im Jahr 1976 nahm Whit Diffie Kontakt zu Martin Hellman auf, um sich mit ihm über Fragen der Verschlüsselung auszutauschen.

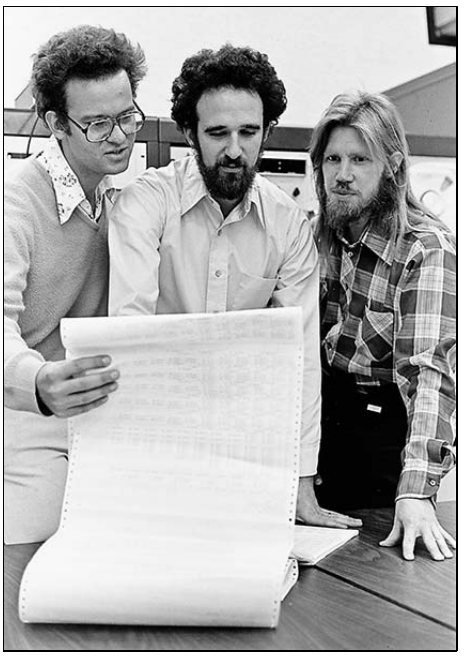


**Bild 1: Whitfield Diffie, Martin Hellman und Ralph Merkle (von rechts nach links) im Jahr 1977 an der Stanford-Universität.**

Foto: Stanford News Service

In Stanford waren sich Hellman und Diffie einig in ihrer Kritik am DES (***D**ata **E**ncryption **S**tandard*), dem gerade eingeführten Standard zur (symmetrischen) Verschlüsselung. Ein wesentlicher Kritikpunkt war die Beschränkung der Schlüssellänge auf 56 Bit, die den Verdacht nährte, dass die staatliche „Abhör- und Verschlüsselungsbehörde" NSA diese Methode mit ihrer überlegenen Rechentechnik und „brute force" gerade noch würde brechen können.

Die Zusammenarbeit zwischen Diffie und Hellman, zu denen 1977 noch Ralph Merkle stieß, erwies sich als sehr fruchtbar. (In der nächsten Folge von *RSA & Co.* werden wir uns mit Ralph Merkle näher beschäftigen.) Es gab aber auch – wie wir noch berichten werden – einige Rückschläge. Die technische Entwicklung führte dazu, dass neue Lösungen für alte Probleme erforderlich wurden; die Erfindung der asymmetrischen Kryptografie lag sozusagen „in der Luft".

Im Jahr 1976 begann mit der Gründung der Firma Apple und der Vorstellung des ersten auch für Privathaushalte erschwinglichen persönlichen Computers, des *Apple I*, ein neues Zeitalter, in dem Informationstechnik für jedermann zugänglich wurde. Schon in den 1960er-Jahren war mit dem ARPAnet der Vorläufer des Internets entwickelt worden, sodass sich auch die Vernetzung der Informatiksysteme abzeichnete.

Ebenfalls 1976 veröffentlichten Diffie und Hellman das bahnbrechende Papier *New Directions in Cryptography*, das mit den folgenden visionären Sätzen beginnt (vgl. Diffie/Hellman, 1978, S. 644):

> We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals. […]
> The development of computer controlled communication networks promises effortless and inexpensive contact between people or computers on opposite sides of the world, replacing most mail and many excursions with telecommunications. For many applications these contacts must be made secure against both eavesdropping and the injection of illegitimate messages.

Was bedeutet *eavesdropping*? In der Literatur zur modernen Kryptografie ist es seit Rivest, Shamir und Adleman – den RSA-Erfindern – üblich geworden, den Kommunikationspartnern die Namen *Alice* und *Bob* zu geben, weil das netter als A und B oder X und Y klingt. Entsprechend bekommt auch der heimliche Lauscher (oder die Lauscherin) einen richtigen Namen, meist *Eve*. Dieser Name soll an die Bezeichnung *eavesdropper* (deutsch: Lauscher) erinnern und hängt etymologisch mit dem Wort *eaves* (deutsch: Dachvorsprung oder Traufe; vgl. Wikipedia – Stichwort „Eavesdropping") zusammen: Der Lauscher steht dicht am Haus, um das im Haus Gesprochene hören zu können (siehe Bild 2; vgl. auch Lautebach, 2015, Seite 77 ff. in diesem Heft). In der Kryptologie versteht man unter einem Lauscher einen passiven Angreifer.

Das Belauschen von Gesprächen erfolgt heutzutage nicht mehr so beschaulich, wie es der niederländische Maler Nicolaes Maes (1634–1693) darstellte und bei YouTube nachempfunden wurde (https://www.youtube.com/watch?v=qBkvVLaR6nw ), sondern automatisiert und vollelektronisch: Die Codes werden von Computern erzeugt und ggf. mit Computern gebrochen – wir kommen weiter unten darauf zurück.

Quelle: The York Project / Dordrechts Museum

**Bild 2: „Die Lauscherin" (1657), Öl auf Leinwand, 92 x 121 cm, von Nicolaes Maes.**

In dem bereits erwähnten Artikel *New Directions in Cryptography* wurde das Konzept des öffentlichen sowie des dazu passenden geheimen privaten Schlüssels entwickelt. Auch wenn die Autoren noch kein funktionierendes Beispiel für ein solches asymmetrisches Krypto-System vorstellen konnten, zeigten sie die enormen Vorteile auf, die ein solches – bis dahin hypothetisches – System haben würde:

- Man würde das Problem der *Schlüsselverteilung*, für die bisher Boten oder andere sichere Kanäle benötigt wurden, einfach mit einer Art Telefonbuch lösen. Dort könnte jeder Nutzer seinen öffentlichen Schlüssel ablegen, sodass er allen zur Verfügung stehen würde. Wenn eine Nachricht vor dem Lauscher geschützt werden soll, besorgt man sich den öffentlichen Schlüssel (public key) des Empfängers und verschlüsselt damit die Nachricht. Nur der rechtmäßige Empfänger ist im Besitz des geheimen Schlüssels und kann daher die Nachricht entschlüsseln, die *Vertraulichkeit* (privacy) der Kommunikation wäre garantiert, sogar gegen neugierige Geheimdienste.
- Das zweite Problem, das durch die asymmetrische Kryptografie gelöst werden würde, ist das der *digitalen Unterschrift*. Der Unterschreibende verschlüsselt einen möglichst eindeutigen Hashwert (deutsch: Streuwert) des jeweiligen Dokuments mit seinem privaten Schlüssel. Dann kann jeder mit dem öffentlichen Schlüssel überprüfen, ob die Unterschrift korrekt ist, wenn der nochmals berechnete Hashwert mit dem entschlüsselten übereinstimmt. Abweichungen können sich nur ergeben, wenn das Dokument unterwegs verändert wurde oder nicht der korrekte private Schlüssel verwendet wurde. Man erhält so eine *authentische*, fälschungssichere Kommunikation.


**Bild 3: Ausschnitt auf dem E-Learning-Programm „Nachrichten-Verschlüsselung und Digitales Signieren".**

Die Idee eines öffentlichen Schlüssels, aus dem der geheime Schlüssel keineswegs berechnet werden kann, ist so zentral, dass sie diesem neuen Gebiet der Kryptografie seinen Namen gab: Public-Key-Kryptografie. Eine anschauliche Demonstration dieser Verfahren ganz ohne Mathematik bietet das E-Learning-Programm *Nachrichten-Verschlüsselung und Digitales Signieren* aus dem Jahr 2001, das seinerzeit vom Institut für Telematik in Trier unter der Leitung von Christoph Meinel, dem heutigen Direktor des Hasso-Plattner-Instituts an der Universität Potsdam, entwickelt wurde (vgl. Institut für Telematik, 2001).

Für die Erzeugung eines solchen Systems mit öffentlichem Schlüssel schlugen Diffie und Hellman das Konzept der *Einweg-Funktion* (engl.: one-way function) vor und als Erweiterung die Einweg-Funktion mit Hintertür oder Falltür, kurz *Falltür-Funktion* (engl.: one-way function with trapdoor – kurz: trapdoor function).

Bei einer solchen Einweg-Funktion ist jeder Funktionswert einfach zu bestimmen, während es praktisch unmöglich sein muss, das Urbild zu einem vorgegebenen Funktionswert zu berechnen – es sei denn, dass es eine Hintertür gibt. Bei Kenntnis dieser Hintertür wird auch die Berechnung des Urbilds einfach (vgl. auch Kardel, 1984; Müller, 2011).

Ein bekanntes Beispiel für eine Einweg-Funktion (ohne Trapdoor) ist das Telefonbuch: Für einen Namen kann man leicht die Telefon-Nummer finden, da die Namen alphabetisch sortiert sind. Umgekehrt ist es schwierig, zu einer gegebenen Nummer den Namen zu finden (wenn man dies per Hand machen muss). Ein weiteres Beispiel ist das Mischen von Farben. Während das Zusammenrühren der Farben einfach ist, ist es schwierig, die Ausgangsfarben aus dem Farbgemisch zu bestimmen (siehe auch weiter unten).

Ein Bild für eine Falltür-Funktion ist der Briefkasten: Es ist einfach, einen Brief einzuwerfen, aber schwer, ihn wieder herauszuholen – es sei denn, man hat den Schlüssel (= Hintertür) zu dem Briefkasten. Ein weiteres, viel verwendetes Beispiel ist ein Vorhängeschloss: Hier kann das Schloss durch einfaches Zudrücken geschlossen werden; zum Öffnen benötigt man dagegen den Schlüssel, der auch in diesem Fall die Hintertür symbolisiert (siehe unten).

Für kryptologische Zwecke benötigt man mathematische Funktionen. Nach Diffie waren schon bald drei Funktionen auf der Kandidatenliste, die später alle eine wichtige Rolle spielen sollten:

1. Ein Kollege aus Stanford schlug das modulare (diskrete) Potenzieren vor, d.h. das Potenzieren in der endlichen (multiplikativen) Gruppe mit den Zahlen $\{1, \ldots, p-1\}$, wobei $p$ eine Primzahl ist und die normale Multiplikation modulo $p$ zu nehmen ist. Diese Gruppe wird üblicherweise mit $\mathbb{Z}_p^*$ bezeichnet (auch Restklassengruppe mod $p$ genannt). Das Potenzieren ist einfach, die Umkehrung – das diskrete Logarithmieren – gilt dagegen als schwierig, sobald $p$ sehr groß ist. Dieses Verfahren wird beim Diffie-Hellman-Schlüsselaustausch und beim Elgamal-Kryptosystem verwendet.
2. Donald („Don") Knuth, ebenfalls aus Stanford, schlug die Multiplikation zweier großer Primzahlen vor, was auch bei sehr großen Zahlen einfach geht, während das Faktorisieren außerordentlich schwierig ist. Diese Einweg-Funktion bildet die Grundlage von RSA sowie des Rabin-Kryptosystems; sie wurde von den RSA-Erfindern aber wohl unabhängig von Don Knuth gewählt.
3. Das Rucksack-Problem (engl.: *knapsack problem*; vgl. auch Baumann, 2000) bildet die Grundlage für das Merkle-Hellman-Kryptosystem (siehe unten). Die grundlegende Idee dazu kam Whit Diffie, als er sich über Verfahren informierte, deren Lösung beweisbar schwierig sind, die sogenannten NP-vollständigen Probleme. Die Idee wurde dann von Ralph Merkle in seiner Dissertation bei Martin Hellman ausgearbeitet. Allerdings konnte dieses System auf der *Crypto82* von Adi Shamir, Len Adleman und anderen spektakulär gebrochen werden (hierauf soll in der nächsten Folge dieser Beitragsserie zurückgekommen werden).

Es ist im Übrigen bis heute mathematisch noch nicht bewiesen, dass die in der Praxis viel verwendeten Funktionen unter 1. und 2. tatsächlich Einweg-Funktionen sind; ein Beweis würde einen Nachweis von $P \neq NP$ zur Folge haben und damit das wichtigste offene Problem der theoretischen Informatik lösen (vgl. auch Niedermeier u.a., 2007), auf dessen Lösung ein Preis von 1 Million Dollar ausgesetzt ist. In der Praxis muss man sich daher zurzeit noch mit der Tatsache anfreunden, dass die Einweg-Eigenschaft in beiden Fällen lediglich plausibel ist und dass weder für das Diskrete-

Logarithmus-Problem noch für die Faktorisierung (bislang) schnelle, effiziente Algorithmen gefunden wurden. Außerdem lässt sich in beiden Fällen die Schwierigkeit der Umkehrung problemlos durch die Schlüsselgröße skalieren.

## Der Diffie-Hellman-Schlüsselaustausch

In dem bereits genannten Artikel *New Directions in Cryptography* von Diffie und Hellman wurde neben der allgemeinen Vorstellung der asymmetrischen Kryptografie (wie oben erwähnt noch ohne Verschlüsselungsverfahren) ein konkretes Verfahren zum Austausch von Geheimnissen (Schlüsseln) über öffentliche Kommunikationskanäle vorgestellt: der Diffie-Hellman-Schlüsseltausch. Da auch Ralph Merkle an der Ausarbeitung dieses Konzepts beteiligt war, wird dieses Protokoll auch Diffie-Hellman-Merkle-Schlüsselaustausch genannt (vgl. Wikipedia – Stichwort „Diffie-Hellman-Schlüsselaustausch"). Der Grundgedanke dieses Protokolls lässt sich auf zwei nicht-mathematischen Wegen verdeutlichen.

Mit der Farbmischung (Beispiel für eine Einweg-Funktion) erläutern wir den Diffie-Hellman-Schlüsseltausch, von dem man auch in zwei kurzen YouTube-Filmen (in englischer Sprache) eine Visualisierung findet: Im ersten Film wird nur die Bestimmung einer gemeinsamen geheimen Farbe erläutert (vgl. Cruise, 2013), im zweiten Film folgt nach diesem ersten noch eine Demonstration des Diffie-Hellman-Protokolls mit kleinen Zahlen (vgl. Cruise, 2012).

Bei uns sollen entsprechend der Tradition wieder Alice und Bob die Kommunikationspartner sein. Man verständigt sich öffentlich auf eine gemeinsame Grundfarbe (in dem Film-Beispiel die Farbe Gelb). Dann wählen Alice und Bob zufällige geheime private Farben (im Beispiel Rot und Blau) und mischen diese jeweils mit der Grundfarbe (es entstehen bei Alice Gelbrot und bei Bob Blassgrün). Diese Farbmischungen werden an den jeweiligen Partner geschickt; Eve kann daraus weder die private Farbe von Alice noch die von Bob bestimmen, aber Alice und Bob können jetzt diesen Mischungen ihre private Farbe hinzufügen. Nach der erneuten Mischung entsteht eine gemeinsame geheime Farbe, die Eve nicht erzeugen kann (im Film-Beispiel ein Braunlila). Wir bemerken, dass hierfür nur eine Einweg-Funktion benötigt wurde, nicht aber eine Falltür-Funktion.

Im zweiten nicht-mathematischen Beispiel haben Alice und Bob jeweils ein Vorhängeschloss und eine gemeinsam genutzte Kiste, die das zu übertragende Geheimnis (z.B. einen Safe-Schlüssel oder ein Passwort) aufnehmen kann. Zuerst packt Alice ihr Geheimnis in die Kiste und verschließt diese mit ihrem Vorhängeschloss. Sie schickt die verschlossene Kiste auf einem unsicheren Weg zu Bob, der mit seinem Vorhängeschloss die Kiste ein zweites Mal verschließt. Dann wandert die Kiste zurück zu Alice, die ihr Vorhängeschloss entfernt. Danach ist die Kiste immer noch mit Bobs Schloss verschlossen, sodass das Geheimnis unbeschadet zu Bob gelangen kann, der zum Schluss sein Schloss öffnet und an das übertragene Geheimnis gelangt.

Diffie und Hellman benutzten für den nach ihnen benannten Schlüsseltausch die diskrete modulare Exponentialfunktion (d.h. das modulare Potenzieren mit ganzen Zahlen für Basis und Exponent), deren Umkehrung bei genügend großem Modul praktisch nicht berechnet werden kann:

1. Alice und Bob vereinbaren gemeinsam eine große Primzahl $p$ (in der gleichen Größenordnung wie bei RSA) und wählen eine Zahl $g$ mit $1 < g < p - 1$, möglichst einen Generator (d.h. eine Primitivwurzel) von $\mathbb{Z}_p^*$. Diese Zahlen können über einen unsicheren Kanal ausgetauscht werden.
2. Alice und Bob wählen im Geheimen jeder eine Zahl und behalten sie für sich ($a$ bzw. $b$).
3. Alice und Bob berechnen jeweils $g^a \pmod p$ bzw. $g^b \pmod p$ und übermitteln das Ergebnis über einen unsicheren Kanal an ihren Partner.
4. Damit erhalten Alice und Bob das gemeinsame Geheimnis $S$
   $S \equiv (g^a)^b \equiv (g^b)^a \equiv g^{ab} \pmod p$.

Wir geben zur Illustration noch ein Rechenbeispiel mit kleinen Zahlen an, die natürlich kryptologisch vollkommen unsicher sind, da der diskrete Logarithmus in diesem Fall durch simples Probieren gefunden werden kann (vgl. Stein, 2011, S.52; wir geben weiter unten ein Beispiel mit realistischeren Zahlen an):

1. $p = 97, g = 5$
2. $a = 31, b = 95$
3. $g^a \equiv 7 \pmod p$, $g^b \equiv 39 \pmod p$
4. $S \equiv (g^a)^b \equiv 14 \pmod p$

Dieses gemeinsame Geheimnis $S$ von Alice und Bob könnte jetzt z.B. als Schlüssel bei einer symmetrischen Verschlüsselung genutzt werden (wenn die angegebenen Zahlen für kryptologische Zwecke nicht viel zu klein wären).

## Das diskrete Potenzieren und seine Umkehrungen

Genau wie beim Potenzieren im Reellen gibt es auch im diskreten Fall zwei wesentlich verschiedene Umkehrungen. Wenn der Exponent fest und die Basis variabel ist, ergibt sich als Umkehrung das Wurzelziehen. In diesem Fall bezeichnet man die Verschlüsselungsfunktionen als *Polynom-Funktionen* (oder *ganzrationale Funktionen*); die Umkehrung bilden die *Wurzelfunktionen*. Diese Funktionen treten beim RSA-Verfahren und beim Verfahren nach Rabin in ihren diskreten Ausprägungen auf.

Wenn umgekehrt die Basis fest bleibt und die Variable im Exponenten steht, ist das Logarithmieren die Umkehrung. Da im diskreten Fall zusätzlich Definitions- und Wertebereich jeweils endliche Teilmengen der Menge $\mathbb{Z}$ der ganzen Zahlen sind, treten einige Besonderheiten auf, die für die Schülerinnen und Schüler neu sind (Modulo-Rechnen).

Der mathematische Hintergrund und die exakte Definitionen sind im Kasten „Diskreter Logarithmus" festgehalten (unten). Im Unterricht wird man sich diesen neuen Sachverhalten mit vielen Beispielen nähern.

Für das Experimentieren mit den diskreten Exponential- und Logarithmus-Funktionen verwenden wir das frei verfügbare Open-Source-Produkt *SageMath* (vormals *Sage* bzw. *SAGE*; ***S**ystem for **A**lgebra and **G**eometry **E**xperimentation*; vgl. Wikipedia – Stichwort „SageMath"), das sich in seiner Syntax weitgehend an PYTHON 2.x orientiert. Darüber hinaus sind in SageMath u.a. sehr viele Funktionen der elementaren Zahlentheorie implementiert, die uns die kostspielige Beschaffung von Computeralgebrasystemen (CAS) wie *Maple* (***ma**thematical mani**p**ulation **l**anguag**e***), *Mathe-*

## Diskreter Logarithmus

Bei einigen wichtigen öffentlichen Verschlüsselungsverfahren wird als Einwegfunktion das Potenzieren in einer endlichen Gruppe verwandt, wobei die Berechnung der Umkehrung, nämlich des diskreten Logarithmus (s.u.), praktisch unmöglich sein sollte. Im Folgenden werden hier einige der dabei wichtigen Begriffsbildungen beschrieben.

**Zyklische (Unter-)Gruppe**

Sei $G$ eine endliche Gruppe; wir wählen (zunächst beliebig) ein $g \in G$ aus und berechnen die Potenzen von $g$. Die von dem Element $g$ erzeugte Gruppe $\langle g\rangle := \{g^i \mid i \in \mathbb{Z}\}$, eine sogenannte *zyklische Gruppe*, ist enthalten in $G$, also eine Untergruppe von $G$. (Es kann auch ein $g_0$ existieren, für das $\langle g_0\rangle$ gleich $G$ ist; in diesem Falle ist $G$ also selbst zyklische Gruppe.)

**Beispiele endlicher zyklischer (Unter-)Gruppen**

Sei $G = \{1, 2, 3, 4\}$ mit der Multiplikation modulo 5 als Verknüpfung, (also die Gruppe $\mathbb{Z}_5^*$). Dann ist $|G| = 4$, und es gibt die zyklischen Untergruppen $\langle 1\rangle = \{1\}$, $\langle 2\rangle = \{2, 4, 3, 1\} = G$ (wegen $2^2 = 4, 2^3 \equiv 3, 2^4 \equiv 1 (\mathrm{mod}\ 5)$) und $\langle 3\rangle = \{3, 4, 2, 1\} = G$ (wegen $3^2 \equiv 4, 3^3 \equiv 2, 3^4 \equiv 1 (\mathrm{mod}\ 5)$ sowie $\langle 4\rangle = \{4, 1\}$ (wegen $4^2 \equiv 1 (\mathrm{mod}\ 5)$). Insbesondere ist $G = \mathbb{Z}_5^*$ selbst zyklisch.

**Ordnung eines Gruppenelements**

Da die Gruppe $G$ als endlich vorausgesetzt wurde, können in der von $g$ erzeugten zyklischen Gruppe $H := \langle g\rangle = \{g^i \mid i \in \mathbb{Z}\}$, einer Untergruppe von $G$, ebenfalls nur endlich viele Elemente sein. Es gibt daher ganze Zahlen $k$ und $l$ mit $g^k = g^l$ und ohne Beschränkung der Allgemeinheit $k > l$, also $g^{k-l} = g^k/g^l = 1$ und $k - l > 0$. Unter allen natürlichen Zahlen $s$ mit $g^s = 1$ gibt es ein kleinstes Element, also eine kleinste positive ganze Zahl $m$ mit $g^m = 1$ $(= g^0)$. Die Zahl $m = o(g)$ heißt die *Ordnung von g*. Für dieses $m$ gilt dann $H = \{g, g^2, g^3, \ldots, g^m\}$ und für die Anzahl der Elemente von $H$ – die sogenannte *Ordnung von H* – also $|H| = m$.

*Anmerkungen*

1. Jedes Element $g$ einer endlichen Gruppe $G$ hat eine endliche Ordnung.
2. Mit $g^{m+1} = g, g^{m+2} = g^2, \ldots$ beginnt die Reihe $g, g^2, g^3, \ldots, g^m$ $(= 1)$ „von Neuem". Bei den Exponenten rechnet man daher modulo $m$.
   Die negativen Potenzen erhält man mittels der Gleichung $g^{-i} = 1 \cdot g^{-i} = g^{m \cdot k - i}$ für $m \cdot k > i$. Es folgt $\langle g\rangle = \{1, g, g^2, \ldots, g^{m-1}\}$ und $|\langle g\rangle| = m$.
3. Im obigen Beispiel $G = \mathbb{Z}_5^*$ haben die Elemente 2 und 3 jeweils die Ordnung 4, das Element 4 die Ordnung 2 und das Element 1 die Ordnung 1.

**Primitives Element**

Falls $G$ selbst zyklisch ist, so wählt man $g$ oft speziell als erzeugendes Element $g_0$ von $G$. Jedes solche Element $g_0$ heißt *primitives Element*. Im Falle $G = \mathbb{Z}_p^*$ spricht man dann von einer „primitiven $(p-1)$-ten Einheitswurzel".

Im Beispiel $\mathbb{Z}_5^*$ sind 2 und 3 also primitive 4-te Einheitswurzeln.

**Diskrete Exponentialfunktion**

Allgemein liefert das Potenzieren von $g$ die *diskrete Exponentialfunktion*, also die Funktion, die jeder ganzen Zahl $a$ mit $0 \le a < m$ $(:= |H|)$ die Potenz $g^a$ zuordnet: $\{0, 1, 2, \ldots, m-1\} \to H$ mit $a \mapsto g^a$.

**Diskrete Logarithmusfunktion**

Bei sehr großen Gruppen $H$ kommen die diskreten Exponentialfunktionen als *Einwegfunktionen* infrage. Beim Problem der Umkehrung sind $G$, $g$ und der Wert von $g^a \in G$ gegeben, und $a$ ist gesucht. Die zugehörige Funktion heißt *diskrete Logarithmusfunktion* und wird mit $\log_g$ bezeichnet. Also:

$\log_g y := x \Leftrightarrow g^x = y$ und $x \in \{0, \ldots, o(g) - 1\}$.

*Anmerkungen*

1. Ist $G \neq H = \langle g\rangle$, so ist $\log_g y$ für $y \in G$ genau dann definiert, wenn $y$ Element von $H$ ist.
2. Die diskreten Exponential- und Logarithmusfunktionen haben endlichen Definitions- und Wertebereich; ansonsten werden sie analog zur reellen Exponential- und Logarithmusfunktion verwendet. Das Adjektiv „diskret" benutzt man hierbei zur besseren Unterscheidung von diesen *reellen* Funktionen.

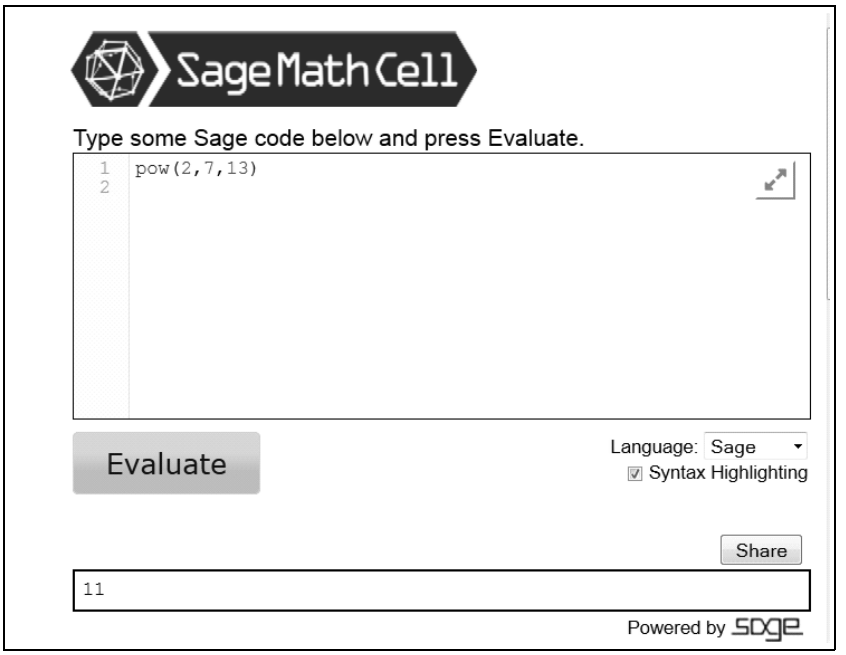



**Bild 4: Der SageMathCell-Server. Im oberen Fenster wird der Sage-Code eingegeben, der nach dem Drücken der „Evaluate"-Schaltfläche ausgeführt wird. In diesem Beispiel wird $2^7 \equiv 11 \pmod{13}$ berechnet (das funktioniert genauso mit einem PYTHON-Interpreter).**

*matica* oder *TI-Nspire* (vormals *Derive*) ersparen. Ein weiteres Argument für SageMath ist, dass keine proprietäre Programmiersprache wie bei den kommerziellen CAS-Systemen zum Einsatz kommt, sondern mit PYTHON eine in der industriellen Praxis, an Forschungseinrichtungen und Universitäten sowie nicht zuletzt an den Schulen seit vielen Jahren stark verbreitete und bewährte Open-Source-Sprache. Von Magnus Lie Hetland ist beispielsweise ein sehr kompakter Intensivkurs für PYTHON 2.x im Internet zu finden, der auch in deutscher Sprache verfügbar ist (vgl. Hetland, 2014). Ansonsten empfiehlt sich das offizielle PYTHON-Tutorial, das vom Schöpfer der Programmiersprache PYTHON, Guido van Rossum, persönlich stammt, dem BDFL des PYTHON-Projekts (***B**enevolent **D**ictator **f**or **L**ife*; deutsch: wohlwollender Diktator auf Lebenszeit; vgl. Python Software Foundation, 1990ff.). Im Jahr 2013 wurde im Übrigen William Stein aufgrund des SageMath-Projekts mit dem *Richard Dimick Jenks Memorial Prize* der Special Interest Group Symbolic and Algebraic Manipulation (SIGSAM) der Association for Computing Machinery (ACM) ausgezeichnet, einem Preis, der alle zwei Jahre für außergewöhnliche Leistungen auf dem Gebiet der Computeralgebra verliehen wird (vgl. SIGSAM, 2013).

Für unsere kleinen Experimente verwenden wir am einfachsten den SageMathCell-Server, der sich auch in HTML-Seiten einbinden lässt, wenn man den Lernenden interaktive Arbeitsblätter zur Verfügung stellen will. Dafür muss nichts installiert werden; man benötigt lediglich eine Internet-Verbindung und einen Browser. Der Aufruf erfolgt mit

https://sagecell.sagemath.org/
oder
http://sagecell.sagemath.org/

Wir betrachten bei den folgenden Beispielen nur die Gruppen $\mathbb{Z}_p^*$, obwohl der Kasten „Diskreter Logarithmus" (siehe vorige Seite) ganz allgemein für zyklische (Unter-)Gruppen formuliert wurde (vgl. auch Schulz u.a., 2015, Seite 103ff. in diesem Heft). Ohne nähere Begründung erinnern wir daran, dass $p$ eine Primzahl sein muss.

Wie bereits erwähnt, enthält $\mathbb{Z}_p^*$ die $(p-1)$ Zahlen $\{1, \ldots, p-1\}$. Die Null muss ausgeschlossen werden, damit sich alle Zahlen aus $\mathbb{Z}_p^*$ invertieren lassen (sonst wäre es keine Gruppe; dieser Ausschluss wird durch den hochgestellten Stern symbolisiert). Die Zahlen oberhalb von $p$ und unterhalb von 0 werden jeweils durch den Rest $r$ mit $0 \leq r < p$ bei der Division durch $p$ ersetzt, der ebenfalls in dieser Menge liegen muss. Damit motiviert sich auch der Name *Restklassengruppe*. Da $p$ und alle Vielfachen von $p$ bei der Division durch $p$ den Rest 0 lassen, müssen auch sie ausgeschlossen werden. Die bekannten Rechenregeln für die ganzen Zahlen übertragen sich ganz natürlich auch auf die Restklassen. Für eine ausführlichere Darstellung verweisen wir z.B. auf die Kapitel 1 bis 3 aus dem Buch von William Stein, das kostenfrei herunterladbar ist (vgl. Stein, 2011; *Anmerkung:* In diesem Buch wird für die Restklassengruppe $\mathbb{Z}_p^*$ die ebenfalls übliche ausführliche Bezeichnung $(\mathbb{Z}/p\mathbb{Z})^*$ verwendet).

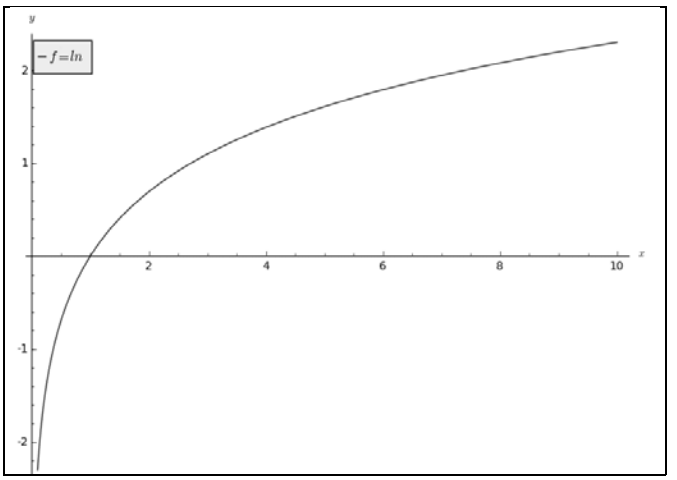



**Bild 5: Die reelle Logarithmus-Funktion zur Basis e (erzeugt mit dem folgenden Sage-Code).**

```
plot(log, 0.1,10, color="blue", axes_labels=['$x$','$y$'],
legend_label='$f=ln$',show_legend=True)
```

Für Pfeile an den Achsen gibt es bei Sage plot leider noch kein Attribut (steht aber seit Längerem auf der SageMath-To-do-Liste).

Nun wollen wir uns der Berechnung der Funktionswerte der diskreten Exponentialfunktion zuwenden. Dafür verwenden wir am besten die bei SageMath eingebaute, auch bei sehr großen Zahlen effiziente PYTHON-Funktion pow mit den drei Argumenten: Basis, Exponent und Modul. Um z. B. $2^7$ mod 13 zu berechnen, gibt man pow(2,7,13) ein und drückt die „Evaluate"-Schaltfläche; das Ergebnis 11 wird im unteren Fenster angezeigt (siehe Bild 4, vorige Seite).

SageMath bietet auch die Möglichkeit, Funktionsgraphen zu plotten. Dies nutzen wir, um die Einweg-Eigenschaft der diskreten Exponentialfunktion plausibel zu machen. Im Bild 5 (vorige Seite) sehen wir die Umkehrung der normalen (reellen) Exponentialfunktion; im Bild 6 den diskreten Logarithmus modulo 53 (vgl. Stein, 2011, S.54, Figure 3.2). William Stein stellt dazu die rhetorische Frage: "Which picture looks easier to predict?"

Das Bild 6 wurde mit folgendem Sage-Code erzeugt:

```
p = 53
g = primitive_root(p)
```

Hiermit erhalten wir einen Generator $g = 2$ für $\mathbb{Z}_{53}^*$ (s. u. und Kasten „Diskreter Logarithmus", Seite 89): $g$ ist also das kleinste Element von $\mathbb{Z}_{53}^*$, das das komplette $\mathbb{Z}_{53}^*$ erzeugen kann.

```
v = sorted([(pow(g,n,p), n) for n in range(p-1)])
```

pow(g,n,p) nimmt nur Werte zwischen 1 und $p$–1 an (auch wenn der Bereich (range) größer wäre); die PYTHON-Funktion range(p–1) liefert die Liste der Funktionswerte [0, …, $p$–2].

Die letzte Zeile erzeugt eine Liste mit den Wertepaaren der Relation (f(n), n), die anschließend nach f(n) sortiert wird, sodass die Umkehrfunktion (der diskrete Logarithmus modulo 53) ausgegeben werden kann:

```
G = plot(point(v, pointsize=50, color='blue',
    legend_label='$n=dlog(x);x=pow(g,n,p)$',
    show_legend=True))
H = plot(line(v, rgbcolor=(0.5,0.5,0.5)))
J = plot(0, 0,p-1, color="white",
    axes_labels=['$pow(g,n,p)$','$n$'])
G + H + J
```

plot() mit einzelnen Punkten (mit oder ohne Line) kennt das Attribut axes_labels nicht, sodass die Dummy-Funktion J eingeführt wurde, um Achsenbeschriftungen zu erzeugen.

Nach diesen Vorbereitungen können wir mit den Experimenten zur diskreten Exponentialfunktion beginnen. Aus Platzgründen nehmen wir in einem weiteren Beispiel einen kleineren Wert für $p$. Auf die Liste der Eingabewerte [0, …, $p$–2] wird jeweils der Funktionsterm der diskreten Exponentialfunktion $g^n$ mod $p$ angewendet ($p$–1 würde nach dem kleinen Satz von Fermat als Funktionswert wieder 1 liefern und damit den Funktionswert von $g^0 = 1$ wiederholen; insgesamt haben wir also, wie es sein muss, $p$–1 Wertepaare):

```
p = 7
g = 3
[(n, pow(g,n,p)) for n in range(p-1)]
```

Dieser PYTHON- bzw. Sage-Code liefert die Ausgabe

```
[(0, 1), (1, 3), (2, 2), (3, 6), (4, 4), (5, 5)].
```

Um die Ausgabe übersichtlicher zu gestalten, lassen wir uns mit

```
[pow(g,n,p) for n in range(p-1)]
```

lediglich die Liste der Funktionswerte ausgeben:

```
[1, 3, 2, 6, 4, 5]
```

Wir erhalten offenbar eine Permutation der Zahlen von 1 bis $p$–1 = 6. Muss das immer so sein? Um dies zu testen, setzen wir $g$ auf den Wert 2 und erhalten die Liste

```
[1, 2, 4, 1, 2, 4]
```

Hier werden nur die Zahlen 1, 2 und 4 ausgeben, dafür zweimal. Das liegt daran, dass $g$ = 2 kein primitives (erzeugendes) Element ist, $g$ = 3 dagegen schon. $g$ = 2 erzeugt lediglich die Untergruppe mit den drei Elementen [2, 4, 1].

Die Verhältnisse sind übersichtlicher, wenn neben $p$ auch ($p$–1)/2 eine Primzahl ist (das ist hier der Fall, da (7–1)/2 = 3 ebenfalls eine Primzahl ist); man spricht dann von *sicheren Primzahlen*. In diesem Fall ist nämlich die Primfaktorzerlegung von $p$–1 bekannt: $p$–1 = 2·((p–1)/2). Dann können die Elemente von $\mathbb{Z}_p^*$ nach dem Satz von Lagrange nur eine Ordnung von einem der vier folgenden Fälle haben: 1, 2, ($p$–1)/2 oder $p$–1 (vgl. Stein, 2011, S.54, Proposition 3.2.4). In unserem Beispiel mit $p$ = 7 und $g$ = 2 hat $g$ die Ordnung 3; $g$ mit $g$ = 3 hat die Ordnung 6. Weitere kleine sichere Primzahlen sind 5, 7, 11, 23, 47 (vgl. Karpfinger/Kiechle, 2010, S.104).

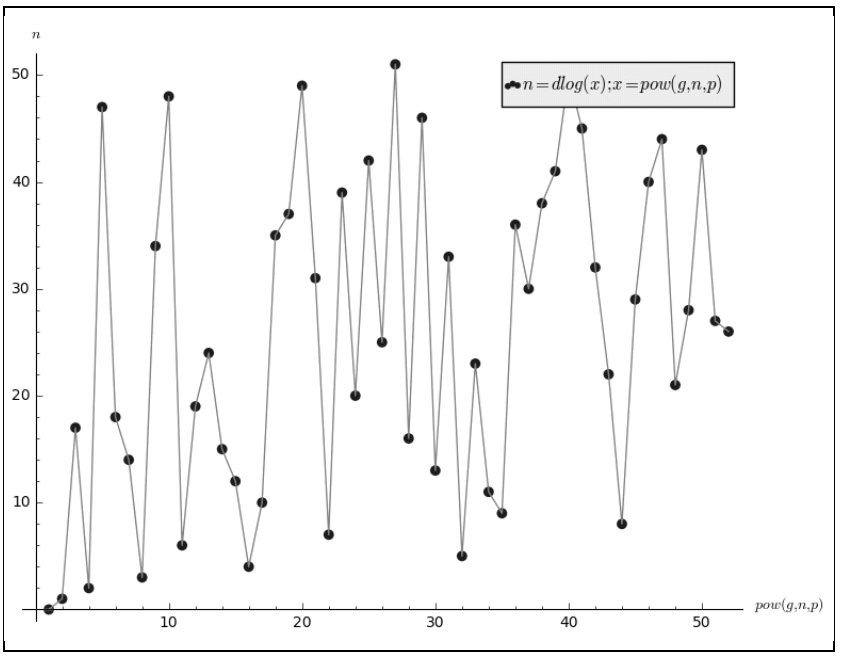



**Bild 6: Die diskrete Logarithmus-Funktion modulo 53.**

Man kann sich mit SageMath zu jeder Primzahl die kleinste zugehörige Primitivwurzel berechnen lassen (das ist die kleinste Zahl, die $\mathbb{Z}_p^*$ erzeugt), im Folgenden nur für alle Primzahlen $p < 20$:

```
for p in primes(20):
    print p, primitive_root(p)
```

liefert die Ausgabe

```
2 1
3 2
5 2
7 3
11 2
13 2
17 3
19 2
```

*Anmerkung:* Die Sage-Funktion primes(n) liefert eine Liste aller Primzahlen kleiner $n$. Der Algorithmus, mit dem die Funktion primitive_root implementiert wurde, ist in Abschnitt 2.5.4 im Buch von Stein beschrieben (vgl. Stein, 2011, S.44). Da für die Berechnung der Primitivwurzel $p$–1 faktorisiert werden muss, ist sie für große $p$ unter Umständen schwierig zu bestimmen.

## Diffie-Hellman mit größeren Zahlen

Wir kommen jetzt zum Diffie-Hellman-Schlüsseltausch zurück und geben ein Beispiel mit großen, eher realistischen Zahlen (vgl. Stein, 2011, S.54f.). Für die Berechnungen verwenden wir wieder SageMath.

Wir beginnen mit einer zufällig gewählten Primzahl

```
q = 9345098309485093845098340961l
```

Leider ist ($q$–1)/2 nicht prim, denn Sage liefert für is_prime((q-1)//2) das Ergebnis False. (In PYTHON 2.x und in SageMath liefert "//" das Ergebnis einer Ganzzahldivision ohne Rest.) Danach wird der Wert von $q$ jeweils um zwei erhöht und geprüft, ob jetzt sowohl $q$ als auch ($q$–1)/2 prim sind. Das gelingt schon nach 6 Versuchen bei der sicheren Primzahl

```
p = 9345098309485093845098340962l3.
```

Wie können wir mit Sage in $\mathbb{Z}_p^*$ rechnen? Mit R = Integers(p) legen wir zunächst fest, dass wir uns in $\mathbb{Z}_p$ bewegen (*Anmerkung für mathematisch Interessierte:* Es handelt sich nicht nur um eine Gruppe, sondern auch um einen Ring. Da $p$ eine Primzahl ist, haben wir es hier sogar mit einem Körper zu tun (vgl. z.B. Wikipedia – Stichwort „Körper (Algebra)"). Wir wählen mit g = R(2) das zweite Ringelement mit dem Wert 2 aus. Um zu testen, ob 2 in $\mathbb{Z}_p^*$ die Ordnung $p$–1 hat und damit Primitivwurzel ist oder nur ($p$–1)/2 (weil $p$ sichere Primzahl ist, käme sonst nur noch die Ordnung 2 infrage), geben wir ein:

```
R = Integers(p)
g = R(2)
g.multiplicative_order()
```

und erhalten nach dem Drücken des „Evaluate"-Schaltfelds die Ordnung des Elements $g$, nämlich die Zahl

```
467254915474254692254917048l1.
```

Wir haben den Verdacht, dass es sich hierbei um ($p$–1)/2 handelt. Mit

```
g.multiplicative_order() == (p-1)//2
```

und der Ausgabe True erhalten wir die Bestätigung, dass 2 wirklich nur die Ordnung ($p$–1)/2 hat und folglich nicht primitiv ist. (*Anmerkung:* Falls 2 primitiv wäre, hätten wir bereits ein erzeugendes Element gefunden und wären fertig.)

Dieses Ergebnis erleichtert nun die Angabe eines Generators $g$ für $\mathbb{Z}_p^*$. Weil 2 also die Ordnung ($p$–1)/2 hat, folgt $2^{(p-1)/2} \equiv 1 \pmod{p}$ und damit

$(-2)^{(p-1)/2} \equiv (-1)^{(p-1)/2} \cdot 2^{(p-1)/2} \equiv (-1) \cdot 1 \not\equiv 1 \pmod{p}$.

–2 hat damit nicht die Ordnung ($p$–1)/2. Aber –2 kann auch nicht die Ordnung 1 oder 2 haben, sodass nur die Ordnung $p$–1 übrig bleibt. Damit ist –2 Generator von $\mathbb{Z}_p^*$. Mit

```
g = Mod(-2, p)
g.multiplicative_order()
```

erhalten wir als Bestätigung

```
9345098309485093845098340962l
```

und somit $p$–1, d.h. $g$ ist tatsächlich ein erzeugendes Element für $\mathbb{Z}_p^*$! (*Anmerkung*: Dieses Verfahren führt immer zu einem erzeugenden Element, wenn $p$ eine genügend große sichere Primzahl ist ($p$>5)!)

Jetzt wählen Alice und Bob ihre geheimen Zufallszahlen $a$ und $b$:

```
a = 8233583624386669568014144030l
b = 1831992237553185917161337918l,
```

die sie sicher verwahren müssen, und berechnen jeweils $A = g^a \pmod{p}$ bzw. $B = g^b \pmod{p}$:

```
A = 1504807415177088427182422539l
B = 4541677627048536979137594499l8.
```

Diese beiden letztgenannten Zahlen können Alice und Bob getrost über einen unsicheren Kanal tauschen, weil es praktisch unmöglich ist, daraus wieder die geheimen Zahlen $a$ und $b$ zu berechnen. Danach kommt der entscheidende Schritt: Beide können jetzt ganz einfach ihr gemeinsames Geheimnis

```
S = 8577140947077052121234673954l
```

mit $(g^a)^b$ bzw. $(g^b)^a \pmod{p}$ berechnen, ohne erneut über eine unsichere Leitung kommunizieren zu müssen:

```
(g^a)^b
8577140947077052121234673954l
(g^b)^a
8577140947077052121234673954l.
```

(*Anmerkung:* Der Potenzoperator ist in PYTHON normalerweise „**", in SageMath kann man abweichend davon auch „^" verwenden – wie in vielen ande-

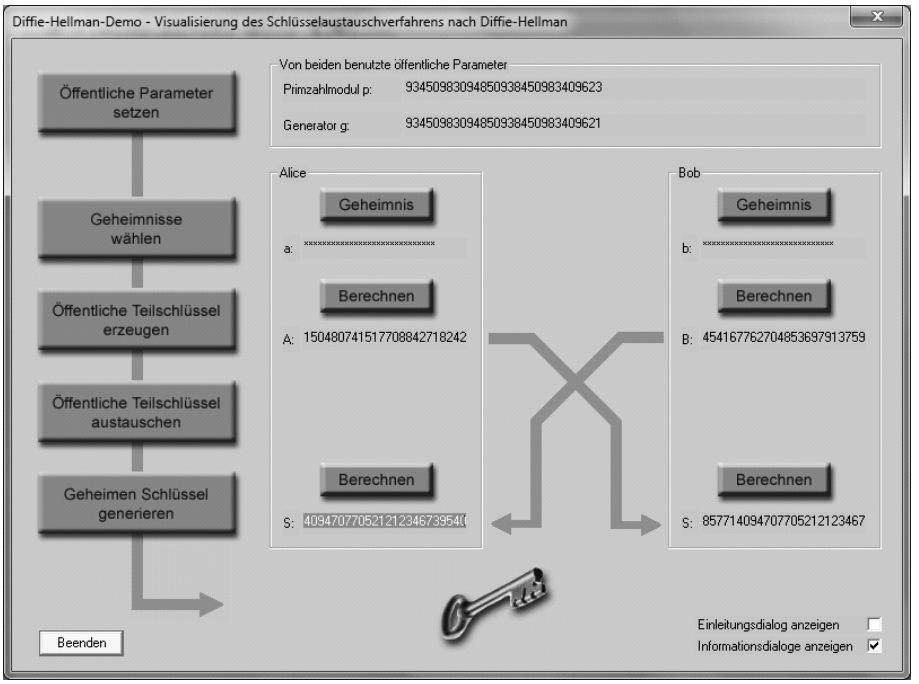



**Bild 7: Visualisierung des Schlüsselaustauschverfahrens nach Diffie-Hellman mit CT1.**

Die Zahlen A, B und S werden nur in Ausschnitten gezeigt, weil die Maske dafür nicht genügend Platz bietet. Man kann in den Zahlenfeldern aber horizontal scrollen. So werden bei dem linken unteren Feld für S nur die letzten Ziffern angezeigt, im rechten die ersten; fünf Ziffern sind jeweils verdeckt.

ren Computeralgebrasystemen (CAS) auch. Da wir uns in $\mathbb{Z}_p^*$ bewegen, muss nicht noch extra modulo $p$ reduziert werden – das geschieht hier automatisch!)

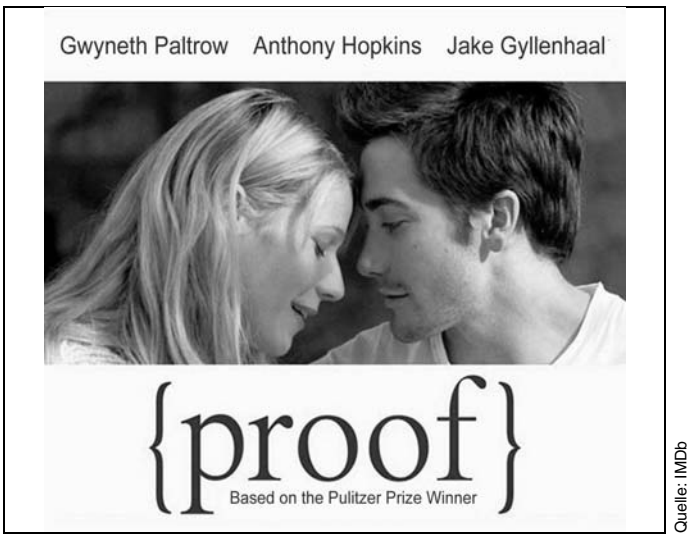
Das Diffie-Hellman-Protokoll zum Schlüsselaustausch über einen unsicheren Kanal kann auch mit CrypTool (CT1 und JCT) visualisiert werden. Im Bild 7 wird die Visualisierung des eben durchgerechneten Beispiels mit CrypTool 1 gezeigt.

Muss man bei Diffie-Hellman sichere Primzahlen verwenden? Nein, es reicht, dass die Ordnung der von $g$ erzeugten Untergruppe groß genug ist. Wenn $p$ sehr groß ist, ist $(p–1)/2$ nur etwa um den Faktor 2 kleiner und damit noch hinreichend groß. Es muss aber auf jeden Fall vermieden werden, dass man in eine Untergruppe mit so wenigen Elementen hineingerät, dass alle Möglichkeiten einfach ausprobiert werden können. (Das ist bei sicheren Primzahlen gewährleistet, vermutlich motiviert sich so ihr Name!) Deshalb ist die Mindestanforderung an $p$, dass $p–1$ einen großen Primteiler besitzt und $g$ so gewählt wird, dass man in der Untergruppe mit einer großen Ordnung operiert.

Nach Karpfinger/Kiechle ist es aufwendig, sehr große sichere Primzahlen zu finden (vgl. Karpfinger/Kiechle, 2010). Bei moderaten Größen von $p$ gelingt dies unter Umständen aber auch recht schnell (siehe das oben angeführte Beispiel; die dort angegebene Primzahl $p$ hat 29 Dezimalstellen bzw. 97 Bit und ist nach heutigen Maßstäben eindeutig zu klein – als aktuelle Mindestgröße gelten 2048 Bit). Eine schöne Aufgabe für die Lernenden wäre es, sichere Primzahlen in dieser Größenordnung mit SageMath zu suchen und so die Aussage von Karpfinger/Kiechle zu verifizieren oder zu widerlegen! Darüber hinaus weiß man bis heute nicht, ob es unendlich viele sichere Primzahlen gibt (vgl. Karpfinger/Kiechle, 2010, S. 104 und S. 143).

In dem Artikel zu sicheren Primzahlen aus der englischsprachigen Wikipedia sind weitere Ausführungen zur kryptografischen Bedeutung dieser speziellen Primzahlen zu finden (vgl. Wikipedia – Stichwort „Safe prime“). Dort ist auch ein interessanter Hinweis enthalten, warum sichere Primzahlen trotz ihres selteneren Auftretens leichter zu finden sind als sogenannte starke Primzahlen (vgl. Wikipedia – Stichwort „Strong prime“) und deshalb in letzter Zeit häufiger verwendet werden:

Safe primes are more time consuming to search for than strong primes, and for this reason they have been less used. However, as computers get faster safe primes are being used more. Finding a random 500-digit safe prime is now quite practical. The problem is that safe primes have the same low density as twin primes. For example, the smallest k such that $2^{25} + k$ is a safe prime is k = 1989, which means that finding it requires testing approximately 1989 numbers for primality. Apart from their low density, they are easier to find than strong primes, in that the programs are much simpler. It is not necessary to attempt the factorization of p – 1. (If p – 1 is difficult to factor, then p is rejected, and p + 2 is tried. This is repeated until p – 1 factors easily. This will happen sooner than p would become a safe prime, on average, because primes p for which p – 1 factors easily are fairly dense.) All of this is made possible by the fact that there are extremely fast probabilistic tests for primality, such as the Miller-Rabin primality test.




Quelle: IMDb

**Bild 8: Ausschnitt aus einem Filmplakat zum US-amerikanischen Film *{proof}* aus dem Jahr 2005.**

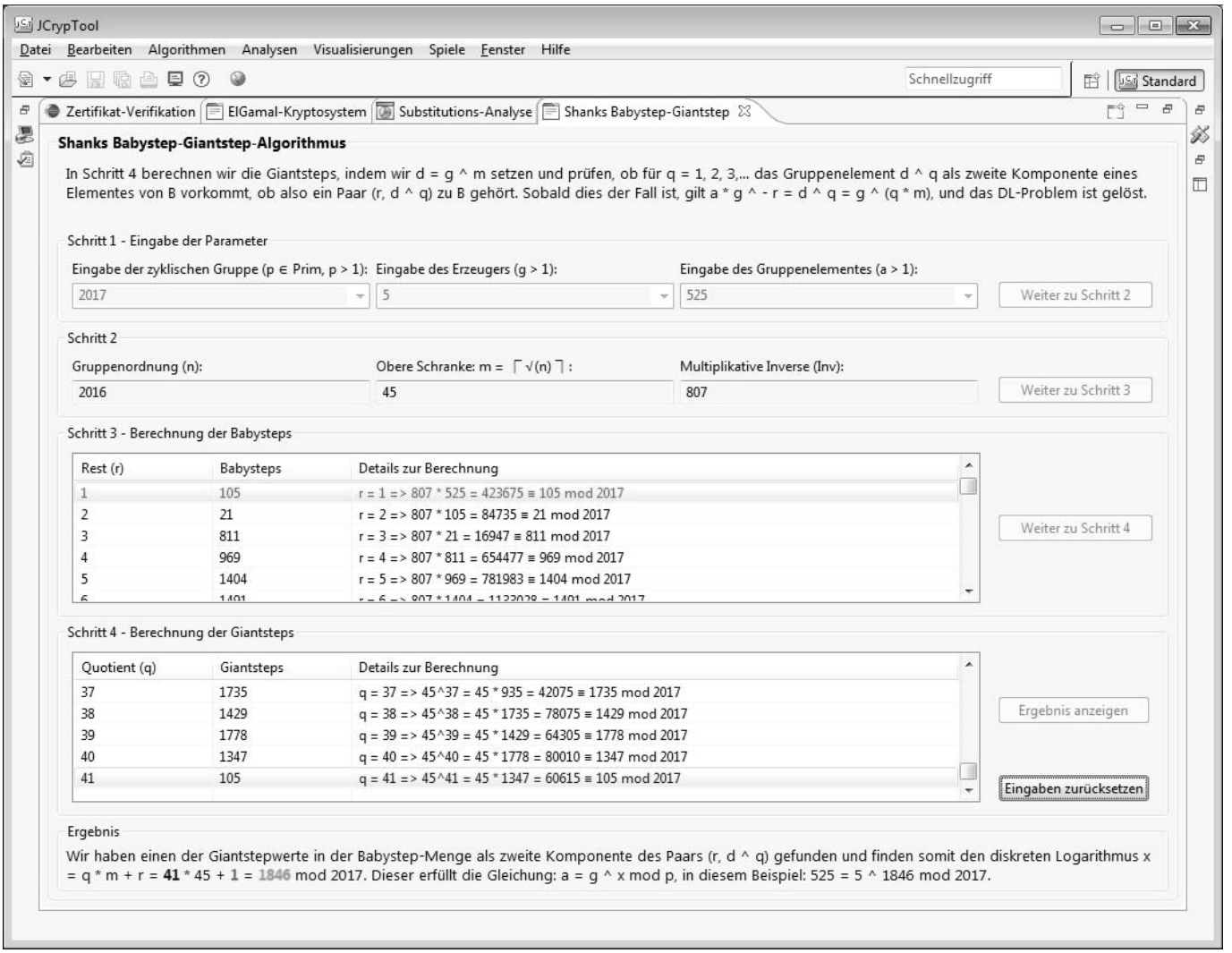



**Bild 9:**
**Babystep-Giantstep mit JCrypTool (JCT).**

Im Übrigen sind die Primzahlen der Form $(p–1)/2$ mit sicherer Primzahl $p$ in der Zahlentheorie schon länger unter dem Namen *Sophie-Germain-Primzahlen* bekannt (vgl. Wikipedia – Stichwort „Sophie Germain prime“); über ihre Dichte gibt es eine Vermutung, die mit der Dichte von Primzahlzwillingen zusammenhängt und auf die berühmten englischen Zahlentheoretiker G.H. Hardy (1877–1947) und J.E. Littlewood (1885–1977) zurückgeht.

Sophie-Germain-Primzahlen werden auch im Theaterstück und dem darauf beruhenden Film *{proof}* (deutscher Titel: *Der Beweis – Liebe zwischen Genie und Wahnsinn*; vgl. IMDb, 2005) erwähnt, der mit prominenter Besetzung ab 2005 in den Kinos lief (siehe Bild 8, vorige Seite).

## Der diskrete Logarithmus (DL)

### Die Sicherheit der modularen Exponentialfunktion

Beim Diffie-Hellman-Verfahren sind der Modul $p$ und das erzeugende Element $g$ öffentlich bekannt, außerdem werden $g^a$ bzw. $g^b$ von Alice bzw. Bob veröffentlicht. Wenn Eve daraus eine der geheimen Zahlen $a$ oder $b$ berechnen kann, hat sie das System „geknackt“, denn jetzt kann auch sie das Geheimnis $S$ berechnen.

Nach Verabredung gilt für Alice' geheime Zahl $g^a \equiv A \bmod p$ und somit $a = \log_g A \bmod p$. Wenn Eve über eine Möglichkeit verfügt, diesen diskreten Logarithmus (DL) zu berechnen, hat sie den Diffie-Hellman-Schlüsseltausch gebrochen. Ob auch die Umkehrung gilt – d.h. wer Diffie-Hellman brechen kann, kann auch den DL bestimmen –, ist zurzeit ebenso unbekannt wie alternative Methoden zum „Knacken“ dieses Systems. Daher konzentrieren sich die Untersuchungen der Sicherheit von Diffie-Hellman auf den DL.

Die simpelste Methode, den DL zu berechnen, ist nacheinander die Elemente $g$, $g^2$, $g^3$, … zu berechnen, bis man auf $g^l \equiv A \bmod p$ gestoßen ist – denn dann ist $l = \log_g A \bmod p$ die gesuchte Umkehrung.

Im Kasten „Der Babystep-Giantstep-Algorithmus“ (siehe nächste Seite) ist exemplarisch ein Algorithmus

zur Berechnung des DL aufgeführt, der eine deutlich bessere Komplexität als der naive Such-Algorithmus hat.

In JCrypTool (JCT), der JAVA-Variante von CrypTool, gibt es eine sehr anschauliche Visualisierung dieses Algorithmus mit einem durchgerechneten Zahlenbeispiel (siehe Bild 9, vorige Seite).

In Karpfinger/Kiechle (2010, Kap. 10) sowie im *CrypTool-Skript* von Bernhard Esslinger u.a. ([11]2013, Abschnitt 5.4.1, S. 218 f.) sind weitere effiziente Algorithmen zur Berechnung des DL aufgeführt, u.a. der Algorithmus von Pohlig-Hellman (vgl. Wikipedia – Stichwort „Pohlig-Hellman-Algorithmus"). Keiner der bekannten DL-Algorithmen besitzt polynomiale Komplexität, sodass die Einweg-Eigenschaft der modularen Exponentialfunktion bislang noch nicht widerlegt wurde – über aktuelle Rekorde bei der Berechnung des DL informiert z.B. die englische Wikipedia unter dem Stichwort „Discrete logarithm records".

Allerdings hat der US-amerikanische Mathematiker und Informatiker Peter W. Shor in seinem viel beachteten Papier zur Quanten-Kryptologie nicht nur einen polynomialen Algorithmus für die Faktorisierung vorgelegt, sondern auch einen entsprechenden Algorithmus für das DL-Problem (vgl. Shor, 1996). Shor spricht in seinem Papier mit Bedacht von einem hypothetischen Quantencomputer, denn die bisherigen Realisierungen sind noch weit davon entfernt, die Sicherheit der in der Praxis verwendeten Einweg-Funktionen tatsächlich infrage zu stellen, obwohl auch die NSA auf diesem Gebiet heftig forscht.

## Der Babystep-Giantstep-Algorithmus

Entwickelt wurde der Babystep-Giantstep-Algorithmus vom US-amerikanischen Mathematiker Daniel Shanks (1917–1996; vgl. Silverman/Tate, 1992, oder Wätjen, 2004). Von 1959 bis zu seinem Tod war er u.a. Mitherausgeber der Zeitschrift *Mathematics of Computation*.

**Aufgabenstellung**

Seien $G$ eine endliche Gruppe, $g \in G$ und $A \in H := \langle g \rangle$. Gesucht ist

$a = \log_g A$, also $a$ mit $A = g^a$.

**Heuristik**

Sei $m$ die Ordnung von $g$ (siehe Kasten „Diskreter Logarithmus", Seite 89). Die natürliche Zahl $w$ mit $w - 1 < \sqrt{m} \leq w$ erfüllt die Gleichung $w^2 \geq m$. Jede natürliche Zahl $x$ mit $x \in \{0, \ldots, m-1\}$ lässt sich durch Division durch $w$ mit Rest in der folgenden Form darstellen: $x = w \cdot j + r$ mit $r \in \{0, \ldots, w-1\}$. Wegen $x < m \leq w^2$ folgt auch $j \in \{0, \ldots, w-1\}$. Daher gilt für $x = a = \log_g A$ mit $a \in \{1, 2, \ldots, m-1\}$ dann $A = g^a = g^{w \cdot j + r}$, was zu

$A \cdot g^{-r} = (g^w)^j$

äquivalent ist.

**Beschreibung des Verfahrens**

Bei der Bestimmung von $a = w \cdot j + r$ sind $r$ und $j$ unbekannt. Von diesen beiden Zahlen steht eine auf der linken und die andere auf der rechten Seite der in der Heuristik hergeleiteten Gleichung $A \cdot g^{-r} = (g^w)^j$; daher betrachtet man beide Seiten zunächst gesondert. Die linke Seite legt die Berechnung der

*Babystep-Liste* $\{A \cdot (g^{-1})^r \mid r = \{0, \ldots, w-1\}$

nahe; bei der Bestimmung der Elemente der (nicht von $a$ abhängenden)

*Giantstep-Liste* $\{1, g^w, \ldots, (g^w)^{w-1}\}$

vergleicht man in jedem Schritt, ob $(g^w)^j$ in der Babystep-Liste vorkommt. In diesem Fall, also für $(g^w)^j = A \cdot g^{-r}$, ist $A = g^{w \cdot j + r}$ und damit $a = \log_g A$ mittels $j$ und $r$ als $a = w \cdot j + r$ bestimmt.

*Anmerkungen*

1. Die Effizienz des viel Speicher verbrauchenden Verfahrens kann durch Anwendung einer Hash-Funktion auf die Einträge der Babystep-Liste verbessert werden.
2. Die Giantstep-Liste lässt sich im Voraus berechnen und sortieren; dann würde man die Vergleiche bei Erstellung der Babystep-Liste vornehmen.

**Komplexität des Algorithmus**

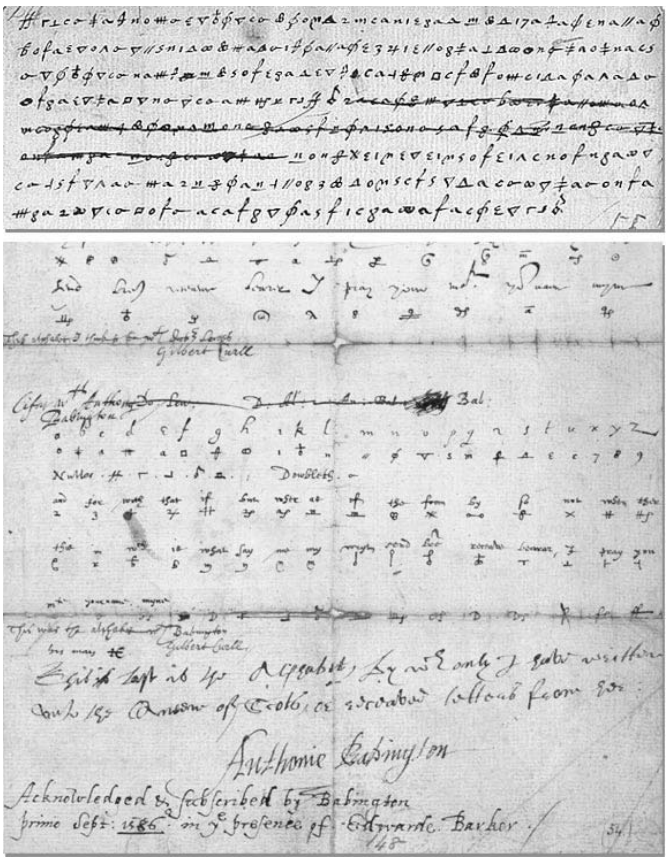
Wie viele Gruppenoperationen und Vergleiche benötigt man bei diesem Algorithmus maximal? Für die Berechnung der Babystep-Liste sind eine Invertierung (von $g$) und (mit $w \approx \sqrt{m}$) auch $w$ Multiplikationen nötig; für die Giantstep-Liste braucht man weniger als $w$ Multiplikationen zur Berechnung von $g^w$ und höchstens $w$ Gruppenoperationen bis zum letzten Element der Liste. In manchen Fällen ist die Invertierung sehr einfach, sodass dann etwa $3w$ (= $O(\sqrt{|H|})$) Gruppenoperationen nötig sind. Weiter sind die Vergleiche (mittels Hashliste als konstant bewertet) und Speicherschritte zu berücksichtigen.

Wenn die auftretenden Zahlen eine Länge von etwa 1000 Bits haben, so braucht man bei der Berechnung der diskreten Exponentialfunktion, also von $g^a$, (mittels „Square and Multiply") ca. 2000 Multiplikationen, zur Berechnung des diskreten Logarithmus mittels des Babystep-Giantstep-Algorithmus aber ca. $\sqrt{2^{1000}} = 2^{500} \approx 10^{150}$ Operationen (vgl. Esslinger u.a., [11]2013). Dies ist Ausdruck der Erfahrungstatsache, dass für große Parameter die diskreten Exponentialfunktionen als Einwegfunktionen aufgefasst werden können. Das gilt auch im Hinblick auf die anderen bisher bekannten Algorithmen.

## The Man in the Middle (MITM)

Der „Mann in der Mitte“ ist ein aktiver Angreifer, der im Gegensatz zur passiven Eve, die nur lauscht, erheblichen Schaden anrichten kann, weil er die Botschaften nicht nur abhören, sondern auch verändern kann. Deshalb hat der namhafte, unabhängige Sicherheitsexperte Bruce Schneier für ihn den Namen *Mallory* eingeführt (von *malicious* = bösartig).

Ob der MITM als bösartig angesehen wird, hängt natürlich vom Standpunkt der Akteure ab. Ein berühmtes historisches Beispiel für einen erfolgreichen Angriff dieser Art ist der – mit einem sogenannten Nomenklator verschlüsselte – Briefwechsel von Maria Stuart mit den jungen katholischen Höflingen um Anthony Babington (vgl. Singh, [10]2011, S. 15 ff.). Ziel der Verschwörer war, die protestantische Königin Elisabeth I. zu töten und die katholische Maria Stuart auf den englischen Thron zu bringen. Da die Verschwörer als Höflinge die Möglichkeit hatten, mit Elisabeth in Kontakt zu kommen, war ihr Geheimdienst alarmiert. (Der Geheimdienst hatte als zusätzliche Sicherheitsmaßnahme



https://de.wikipedia.org/wiki/Babington-Verschw%C3%B6rung
https://de.wikipedia.org/wiki/Nomenklator_(Kryptologie)

**Bild 10: Der im Auftrag von Sir Francis Walsingham veränderte Brief, der angeblich direkt von Maria Stuart kam und damit zur Hinrichtung der Verschwörer und Maria Stuarts führte. Der untere Teil der Abbildung zeigt den verwendeten Schlüssel (einen sogenannten Nomenklator).**

Quelle: DestroyADWare.com – Anti-Spyware Anti-Adware Anti-Malware

**Bild 11: MITM-Angriff.**

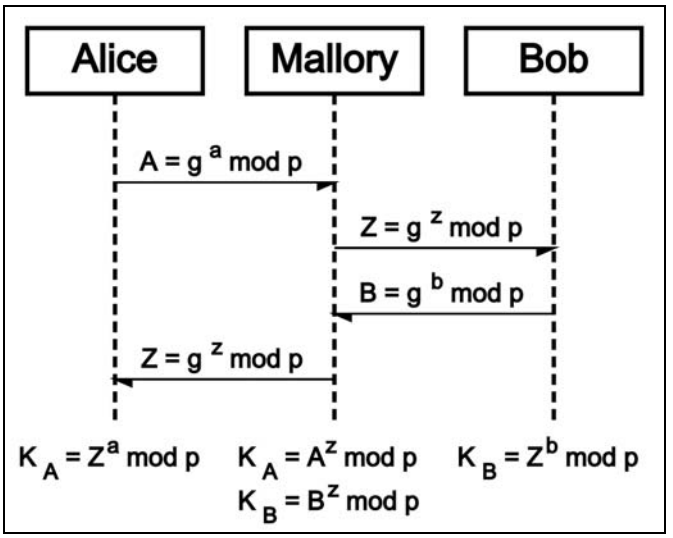
einen Doppelagenten in die Verschwörergruppe eingeschleust.). Der Briefwechsel wurde im Auftrag von Sir Francis Walsingham, dem Sicherheitsminister und engen Mitarbeiter von Königin Elisabeth, abgefangen, von seinem Mitarbeiter Thomas Phelippes dechiffriert und zum Teil erst in veränderter Form an Babington weitergeleitet. Mit diesen Veränderungen versuchte Walsingham die Verschwörer dazu zu bringen, ihre Mitverschwörer preiszugeben (siehe Bild 10).

Schon bald nach der Veröffentlichung des Diffie-Hellman-Schlüsseltauschs wurde erkannt, dass auch dieses Verfahren durch MITM, also durch Mallory, angreifbar ist.

Im Bild 11 wird das Prinzip des MITM-Angriffs verdeutlicht: Links sitzt Alice (schwarz) und denkt, sie kommuniziert mit Bob (weiß, rechts), Bob denkt *vice versa*, entsprechend umgekehrt. In Wirklichkeit hat sich aber Mallory (vorne) eingeschaltet und kann so die komplette Kommunikation abfangen und nach Belieben auch verändern – *worst case* für die unwissenden Opfer Alice und Bob!

Im konkreten Fall des Diffie-Hellman-Schlüsseltauschs funktioniert MITM wie in Bild 12 (nächste Seite) verdeutlicht wird.

Da $g$ und $p$ öffentlich bekannt sind und Mallory die direkte Kommunikation zwischen Alice und Bob unterbindet, kann Mallory statt der von Alice bzw. Bob berechneten Werte $A$ bzw. $B$ beiden den mit seiner eigenen „Geheimzahl“ $z$ berechneten Funktionswert $Z$ unterschieben und daraus die jeweiligen Geheimnisse $K_A$ und $K_B$ berechnen, mit denen er mit Alice und Bob verschlüsselt kommuniziert. Dazu muss bei beiden die Illusion auf-




Quelle: Wikipedia / Stefan Birkner

**Bild 12: MITM-Angriff auf den Diffie-Hellman-Schlüsseltausch.**

recht erhalten bleiben, dass sie direkt miteinander kommunizieren. Wenn Mallory eine Botschaft verpasst und diese direkt zu Alice oder Bob gelangt, könnte er auffliegen. Für einen erfolgreichen MITM-Angriff muss man nicht einmal das Netz komplett kontrollieren, sondern nur bestimmte Verteilpunkte. Um diese Art von Angriffen abzuwehren, sollte man die übermittelten Informationen authentifizieren – entweder durch Signierung oder durch einen ***M**essage **A**uthentication **C**ode* (MAC; siehe auch Lautebach, 2015, Seite 77 ff. in diesem Heft).

Bruce Schneier hat in einem Artikel für *The Guardian* offenbart (vgl. Schneier, 2013), dass die NSA – nach den von Ed Snowden veröffentlichten Dokumenten – MITM-Attacken verwendet, um ausgewählte Zielpersonen zu überwachen, die den TOR-Anonymisierungsdienst verwenden (siehe Bild 13).

Foto: Trevor Paglen / dpa

**Bild 13: Tag und Nacht aktiv – die NSA-Zentrale in Fort Meade im Jahr 2013. Bis zum Zeitpunkt der Veröffentlichung dieser Aufnahme war es nur gestattet, Fotos der NSA-Zentrale ausschließlich von der NSA selbst zu verwenden.**

Obwohl Diffie-Hellman das älteste Protokoll aus dem Bereich der asymmetrischen Kryptografie ist, wird es heute noch – ggf. erweitert und modifiziert – in wichtigen Anwendungen verwendet:

- Der Digitale Signatur Algorithmus (***D**igital **S**ignature **A**lgorithm* – DSA) ist ein Standard, der ebenfalls auf der Schwierigkeit beruht, den diskreten Logarithmus in endlichen Körpern zu berechnen (siehe weiter unten).
- Das Diffie-Hellman-Protokoll wird im OpenSSH-Protokoll Version 2 verwendet (vgl. Stein, 2011, S. 50).
- Das Off-The-Record-Protokoll (OTR) verwendet intensiv Diffie-Hellman (siehe Lautebach, 2015, Seite 77 ff. in diesem Heft).

Die vielleicht wichtigsten Weiterentwicklungen des Diffie-Hellman-Schlüsseltauschs sind aber das Elgamal-Verschlüsselungsverfahren sowie das Elgamal-Signaturverfahren. Beide wurden 1985 unter dem Titel *A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms* veröffentlicht (vgl. Elgamal, 1985).

## Beiträge von Taher Elgamal zur Kryptografie

Taher Elgamal (siehe Bild 14) wurde 1955 in Ägypten geboren und wuchs in Kairo auf; sein Vater war ein hoher Regierungsbeamter. Als Kind war er von Zahlen begeistert. Er studierte Elektrotechnik an der Universität Kairo und legte dort 1977 seinen Bachelor ab. Anschließend ging er in die USA an die Universität Stanford und arbeitete bei Martin Hellman im Bereich der Kryptologie. Dort schloss er sein Studium mit dem Master of Science und der Promotion ab. Inzwischen ist er amerikanischer Staatsbürger und erfolgreicher Unternehmer (vgl. Wikipedia – Stichwort „Taher Elgamal"). *Anmerkung:* Hinsichtlich der Schreibweise seines Nachnamens „Elgamal" (gesprochen: al-Dschamal) gibt es einige Differenzen: Elgamal ist die in der englischsprachigen Literatur übliche Schreibweise, die er selbst bevorzugt. In der deutschen Literatur ist jedoch noch häufig „ElGamal" zu finden.

**Bild 14: Taher Elgamal im Jahr 2010.**

Foto: Wikipedia / Alexander Klink

Von 1995 bis 1997 arbeitete Elgamal als Chef-Wissenschaftler für Netscape und entwickelte u.a. den ***Di**gital **S**ignature **A**lgorithm* (DSA), das Kreditkarten-Zahlungssystem SET sowie das SSL-Protokoll. Seit einiger Zeit ist er einer der Direktoren von RSA Security, einem Unternehmen, das ursprünglich zur Vermarktung des RSA-Kryptosystems gegründet wurde, inzwischen aber als weitgehend von der NSA kompromittiert gelten muss (vgl. Bussemer/Müller, 2014).

### Das Verschlüsselungsverfahren von Elgamal

Das *Verschlüsselungsverfahren von Elgamal* ist nichts anderes als eine Diffie-Hellman-Schlüsselvereinbarung, die – wie oben ausgeführt – zu einem gemeinsamen Schlüssel $S$ führt, mit dem anschließend eine Nachricht $m$ multiplikativ verschlüsselt wird. Wir orientieren uns dabei an der sehr eleganten Darstellung des Verfahrens von Hans Werner Lang (vgl. Lang, 2015b). Um Wiederholungen gering zu halten, führen wir unser oben durchgerechnetes Beispiel fort.

Wir hatten die sichere Primzahl

```
p = 9345098309485093845098340962 3
```

und den Generator `g = Mod(-2, p)` bestimmt. Alice und Bob hatten jeweils (geheime) Zufallszahlen $a$ bzw. $b < p–1$ gewählt und daraus durch modulares Potenzieren die (öffentlichen) Zahlen $A \equiv g^a \pmod p$ bzw. $B \equiv g^b \pmod p$ berechnet. Damit konnten beide den gemeinsamen geheimen Schlüssel $S \equiv A^b \pmod p \equiv B^a \pmod p \equiv 85771409470770521212346739540$ berechnen. Der öffentliche Schlüssel von Alice ist $(p, g, A)$, der geheime private Schlüssel ist $a$. Entsprechend ist der öffentliche Schlüssel von Bob $(p, g, B)$ und der geheime $b$.

Für das Folgende gehen wir davon aus, dass Bob Alice eine geheime Nachricht $m$ schicken will. Dafür berechnet Bob $c \equiv S{\cdot}m \pmod p$ und schickt das Chiffrat an Alice. Alice kann diese Nachricht entschlüsseln, indem sie $S^{-1} \equiv B^{-a} \pmod p \equiv B^{p-1-a} \pmod p$ und damit $S^{-1}{\cdot}c \pmod p \equiv S^{-1}{\cdot}S{\cdot}m \pmod p \equiv m \pmod p$ berechnet.

Als Nachricht verwendet Bob das Wort „ElGamal“, das er nach folgendem einfachen Verfahren in eine Zahl umwandelt: Jeder Buchstabe wird durch die zweiziffrige Stellung im Alphabet ersetzt, also durch 05 12 07 01 13 01 12. Diese Botschaft wird also zu $m$ = 05120701130112. Die Berechnung des Chiffrats $c$ sowie die Dechiffrierung zur Überprüfung der Korrektheit führen wir wieder mit SageMath durch:

```
p = 9345098309485093845098340962 3
R = Integers(p)
g = R(-2); g
```

Hiermit legen wir (wie oben) fest, dass wir in $\mathbb{Z}_p^*$ rechnen und damit automatisch modulo $p$ reduziert wird.

```
a = 82335836243866695680141440300
b = 18319922375531859171613379181
A = g^a
B = g^b
A^b == B^a
```

Die (zufälligen) Geheimzahlen $a$ und $b$ sowie die öffentlichen (Teil-)Schlüssel von Alice und Bob $A$ und $B$ werden ebenso wie oben bestimmt; die letzte Zeile liefert bei der Evaluation `True` und stellt so sicher, dass sich kein Übertragungsfehler eingeschlichen hat. (*Anmerkung:* In PYTHON (und damit auch in SageMath) wird Groß- und Kleinschreibung unterschieden; somit sind $a$ und $A$ zwei verschiedene Variablen).

Als Schlüssel wählt Bob `S = A^b` und verschlüsselt damit seine Nachricht:

```
m = 05120701130112
c = S*m
```

Bob erhält als Chiffrat
111207926761003437973719056 70. Mit

```
m == S^-1*c
```

und der Ausgabe `True` kann er überprüfen, dass die verschlüsselte Nachricht wieder korrekt entschlüsselt wurde; analog kann Alice die Nachricht entschlüsseln.

Die Sicherheit des Elgamal-Verschlüsselungsverfahrens hängt einerseits an der Schwierigkeit, den diskreten Logarithmus zu bestimmen, also der (praktischen) Unmöglichkeit, $a$ und $b$ aus $A$ und $B$ zu bestimmen. Andererseits könnte der gemeinsame, nur Alice und Bob bekannte Schlüssel $S$ von Eve oder Mallory berechnet werden, wenn es ihnen gelänge, eine Known-Plaintext-Attacke durchzuführen, wenn sie also ein zusammengehöriges Paar $m$ und $c$ erbeuten könnten. Dann könnten sie nämlich mit $S = c{\cdot}m^{-1} \pmod p$ den Schlüssel berechnen. Deshalb ist es wichtig, den Schlüssel $S$ jeweils nur ein einziges Mal zu verwenden. Auch in dem Fall, dass $m$ größer als $p$ ist und $m$ deshalb in Teilblöcke zerlegt werden muss, die jeweils einzeln verschlüsselt werden, muss für jeden Teilblock ein neuer Schlüssel bestimmt werden.

Aus diesem Grund nimmt Bob für die Bestimmung des Chiffrats in der Praxis nicht seinen geheimen Schlüssel $b$, weil der dann für die weitere Verwendung „verbrannt“ wäre. Er bestimmt vielmehr jeweils ein (neues) zufälliges $r$, berechnet damit $R = g^r \pmod p$ und sowie $c \equiv A^r{\cdot}m \pmod p$ ($A$ ist der öffentliche Schlüssel von Alice) und sendet $(R, c)$ an Alice, die dann $R^{-a}{\cdot}c \pmod p$ berechnen kann ($a$ ist der private Schlüssel von Alice). Das ist aber wegen $R^{-a}{\cdot}c \equiv g^{-ra}{\cdot}A^r{\cdot}m \equiv g^{-ra}{\cdot}g^{ar}{\cdot}m \equiv m \pmod p$ gerade die von Bob übermittelte Nachricht.

Leider ist der Rechenaufwand für die Elgamal-Verschlüsselung höher als für RSA, weswegen es als asymmetrisches Kryptosystem mit der multiplikativen Restklassengruppe $\mathbb{Z}_p^*$ kaum verwendet wird. Da RSA auf einer elliptischen Kurve nicht funktioniert, aber der diskrete Logarithmus existiert und schwer zu berechnen ist, wird dort Elgamal als das asymmetrische System der Wahl verwendet. Im Kasten „Das verallgemeinerte Elgamal-System“ (siehe nächste Seite) ist das verallgemeinerte Elgamal-System dargestellt, das von den Gegebenheiten der Restklassengruppe abstrahiert und allgemein in endlichen, dafür geeigneten Gruppen anwendbar ist.

### Das Elgamal-Signatur-Verfahren

Das *Elgamal-Signatur-Verfahren* bildet die Grundlage für das standardisierte DSA-Schema. Die Schlüssel-

erzeugung erfolgt wie oben beschrieben; die Schlüsselpaare können für beide Protokolle verwendet werden. Wir gehen nun davon aus, dass Alice ihre Botschaft $m$ signieren will; $p$, $g$, $A$ und $a$ haben die gleiche Bedeutung wie oben definiert. Sie benötigt dafür eine kollisionsresistente Hashfunktion $H$ (vgl. Lang, 2015a), die auch Bob bekannt ist.

## Das verallgemeinerte Elgamal-System

**Wahl des Rahmens**

▷ Seien $G$ eine endliche Gruppe und $g \in G$ derart, dass die Berechnung des diskreten Logarithmus $\log_g$ in $H := \langle g \rangle$ (außer in speziellen Fällen wie $\log_g 1, \log_g g, \ldots$) praktisch unmöglich ist.
Das *Elgamal-Verfahren im engeren Sinne* benutzt als $G$ die multiplikative Gruppe eines endlichen Körpers. (Im Hinblick auf den Schulunterricht vgl. z. B. Baumann, 1996, S. 56 ff.)

▷ Sei ferner $a$ eine natürliche Zahl mit $1 < a < |H|$ und

▷ $A := g^a$.

**Öffentlicher Schlüssel**

$g, A$ (für $A := g^a$).

**Privater Schlüssel**

$a$ ($= \log_g A$).

**Verschlüsselung des Klartextes m ∈ G**

mit einem zufällig gewählten $r$ (der Algorithmus ist damit nicht deterministisch) mit $2 < r < |H| - 1$ und hoffentlich nicht aus $R := g^r$ berechenbarem Wert $r = \log_g R$:
$E(m) := (R, A^r \cdot m)$.

**Entschlüsselung**

bei Kenntnis des privaten Schlüssels:
$D(x, y) := (x^a)^{-1} \cdot y$.

**Verifizierung**

Führt die Entschlüsselung zum ursprünglichen Klartext? Ja, denn es gilt:
$D(E(m)) = D(R, A^r \cdot m) = ((g^r)^a)^{-1} A^r \cdot m = g^{-ar} (g^a)^r \cdot m = m$.

*Anmerkungen*

1. Die Sicherheit des Systems beruht darauf, dass aus $A = g^a$ und $R = g^r$ weder $a$ ($= \log_g A$ – und damit $(x^a)^{-1}$) noch $r$ ($= \log_g R$ – und damit $m = (A^r)^{-1} \cdot A^r \cdot m$) berechnet werden können.
2. Man beachte, dass ein bei einer Übertragung verwendetes $r$ nicht noch einmal benutzt werden sollte (man siehe dazu den Text dieses Beitrags außerhalb des Kastens!).

Alice bestimmt eine Zufallszahl $r$ mit $0 < r < p-1$, sodass $r^{-1} \pmod{(p-1)}$ existiert (dazu muss ggT $(r, p-1) = 1$ gelten). Dann berechnet sie

$$R \equiv g^r \pmod p \text{ sowie } S \equiv (H(m) - a \cdot R) r^{-1} \pmod{(p-1)}. \quad (*)$$

Wenn $S = 0$ gilt, werden die Schritte wiederholt (d. h., eine neue Zufallszahl $r$' erzeugt).

Das Paar $(R, S)$ ist die digitale Signatur von $m$.

Bob überprüft, ob

1. $0 < R < p$ und
2. $0 < S < p-1$ gilt. Außerdem muss
3. $g^{H(m)} \equiv A^R R^S \pmod p$ gelten,

denn aus (*) ergibt sich $H(m) \equiv Sr + aR \pmod{(p-1)}$ und damit $H(m) = aR + rS + (p-1)k$ für ein geeignetes $k$.

Es folgt dann (unter Verwendung des kleinen Satzes von Fermat):

$$g^{H(m)} \equiv (g^a)^R \cdot (g^r)^S \cdot (g^{p-1})^k \equiv A^R \cdot R^S \cdot 1 \equiv A^R R^S \pmod p.$$

Falls die Bedingungen 1. bis 3. erfüllt sind, akzeptiert Bob Alice' Signatur, andernfalls weist er sie zurück (vgl. Wikipedia – Stichwort „Elgamal-Signaturverfahren"). Auch dieses Signaturverfahren funktioniert sowohl in der Restklassengruppe $\mathbb{Z}_p^*$ (wie eben gezeigt) als auch (in modifizierter Form) auf elliptischen Kurven (ECDSA; vgl. auch Wikipedia – Stichwort „Elliptic Curve DSA"). ECDSA ist in OpenSSH sowie in OpenSSL implementiert.

Generell besteht bei der Elliptischen-Kurven-Kryptografie das Problem, dass sehr viele verschiedene Kurven und Parameter möglich sind. Die Auswahl erfolgte durch das US-amerikanische NIST (***N**ational **I**nstitute of **S**tandards and **T**echnology*) nach Beratung durch die NSA. Unabhängige Krypto-Experten wie die deutsche Kryptologin Tanja Lange raten deshalb dazu, neue elliptische Kurven zu suchen, zu erforschen und ggf. zu benutzen (vgl. Ermert, 2014).

Foto: Wikipedia / Dan Spisak

**Bild 15: Die Kryptologen Adi Shamir, Ron Rivest, Marty Hellman und Whit Diffie (v. r. n. l.) – „Four biggest names in Crypto on stage", mit dem Großbild von Whit Diffie (auf der RSA-Konferenz 2008; ganz links: Burt Kaliski als Moderator).**

## Fazit und Ausblick

Whit Diffie, Marty Hellman, Ralph Merkle, Taher Elgamal (alle ehemals Stanford University und Schöpfer der auf dem diskreten Logarithmus beruhenden Kryptosysteme) und Ron Rivest, Adi Shamir, Len Adleman (alle ehemals MIT und Erfinder des RSA-Systems) sind heute noch gefragte Krypto-Experten, mit denen sich Konferenzen zum Thema *Computersicherheit* gerne schmücken (siehe Bild 15, vorige Seite). Rein äußerlich ist Whit Diffie mit seinen inzwischen weißen langen Haaren und dem Vollbart die ungewöhnlichste Erscheinung in dieser Reihe berühmter Kryptologen: Die gepflegte Frisur im Stil der 1960er-Jahre bildet einen interessanten Kontrast zum eleganten dreiteiligen Anzug mit rotem Hemd und auffälliger Krawatte.

In dieser Folge ging es in besonderer Weise um Diffie und Hellman, die durch den nach ihnen benannten Schlüsseltausch unsterblich geworden sind und ohne die das Internet in der heutigen Form undenkbar ist. Von Hellman stammt die treffende Bemerkung: "When a lock icon appears at the bottom of your browser, it's using public key cryptography" (zitiert nach Blackman, 2010).

Angesichts des von Edward Snowden aufgedeckten Geheimdienst-Skandals ist es angebracht, auf das ausdauernde Engagement von Diffie und Hellman für den Schutz der Privatsphäre hinzuweisen. So erhielten sie bereits im Jahr 1994 den *Pioneer Award* der EFF (***E**lectronic **F**rontier **F**oundation*). Die EFF wurde im Jahr 1990 gegründet und veranstaltet seit 1991 jedes Jahr eine *Conference on Computers, Freedom, and Privacy*. Das Wort „Privacy" wird mit „Datenschutz" nur unzureichend übersetzt, gemeint ist der Schutz der Privatsphäre – und der wird durch die anlasslose Massenüberwachung (nicht nur) durch die NSA massiv durchlöchert. Die EFF ist seit dem Jahr 2000 eine offizielle Untergliederung der ACM, dem US-amerikanischen Pendant zur deutschen Gesellschaft für Informatik.

Zur Begründung der Ehrung führte die Jury damals aus (vgl. EFF, 1994):

> Whitfield Diffie of Sun Microsystems and Martin Hellman of Stanford University are the persons chiefly responsible for public-key encryption. In a period in this country's history when the government, and in particular the National Security Agency, had a near-monopoly on encryption technology, Diffie and Hellman developed public-key encryption, a technology that enhances the ability of individuals to keep their communications private and that eliminates the reliance of individuals on third parties to ensure the authenticity of communications. One implementation of Diffie and Hellman's work, Pretty Good Privacy, is a worldwide standard in public-key encryption. Virtually every current public-policy debate on encryption has been profoundly shaped by Diffie and Hellman's work.

Angesichts des NSA-Skandals hat die EFF eine Kampagne gegen die Massenüberwachung gestartet. Auf der Seite https://www.eff.org/nsa-spying werden die Besucher aufgefordert, eine Gesetzesinitiative zur Beendigung der anlasslosen Überwachung zu unterstützen bzw. gegen deren Verwässerung zu opponieren; im Bild 16 wird das Logo dieser Kampagne wiedergegeben.


https://www.eff.org/nsa-spying

**Bild 16: Eine leicht (?) veränderte Version des NSA-Wappens auf den Seiten der Electronic Frontier Foundation.**

Das Original des Wappens findet sich auf den offiziellen Seiten der National Security Agency (http://www.nsa.gov/) und zeigt einen Adler, der in seinen Krallen einen Schlüssel trägt. Die Zeichnung hier spielt dagegen auf den US-amerikanischen Begriff für Abhören an: „wiretapping".

In der nächsten Folge werden wir noch einmal zu den Anfängen der Public-Key-Kryptografie zurückkehren und von den Verdiensten und Niederlagen Ralph Merkles berichten. Außerdem werden wir ebenso über das britische Pendant zur NSA, das ***G**overnment **C**ommunications **H**eadquarters* (GCHQ, deutsch: Regierungskommunikationszentrale) und dessen Rolle bei der Erfindung der asymmetrischen Kryptografie informieren. Darüber hinaus beabsichtigen wir, die Geschichte und die Bedeutung der E-Mail-Verschlüsselung mit PGP bzw. S/MIME zu behandeln.

Helmut Witten
Brandenburgische Straße 23
10707 Berlin

E-Mail: helmut@witten-berlin.de

Prof. Dr. Ralph-Hardo Schulz
Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Mathematik
Arnimallee 3
14195 Berlin

E-Mail: schulz@math.fu-berlin.de

Prof. Bernhard Esslinger
Universität Siegen
Institut für Wirtschaftsinformatik
Hölderlinstraße 3
57076 Siegen

E-Mail: bernhard.esslinger@uni-siegen.de

## Literatur und Internetquellen


Baumann, R.: Informationssicherheit durch kryptologische Verfahren – Vorschläge für den Unterricht. In: LOG IN, 16. Jg. (1996), Heft 5/6, S. 52–61.

Baumann, R.: Das Rucksackproblem – Informatische und kryptologische Aspekte. In: LOG IN, 20. Jg. (2000), Heft 2, S. 47–52.

Blackman, Chr.: Stanford encryption pioneer who risked career becomes Hamming Medalist. Stanford Report, 10. 02.2010.
http://news.stanford.edu/news/2010/february8/hellman-encryption-medal-021010.html

Bussemer, P.; Müller, J.: Was noch sicher ist – Ein Überblick über kryptografische Algorithmen und deren mögliche Probleme. In: LOG IN, 14. Jg. (2014), Nr. 178/179, S. 82–90.

Cruise, B.: Diffie-Hellman Key Exchange. 30.07.2012.
https://www.youtube.com/watch?v=YEBfamv-_do

Cruise, B.: Diffie-Hellman Key Exchange (part 1). 21.02.2013.
https://www.youtube.com/watch?v=ko62sibi668

CrypTool 1 (CT 1):
http://www.cryptool.org/de/cryptool1

CrypTool 2 (CT 2):
http://www.cryptool.org/de/cryptool2

Diffie, W.: The First Ten Years of Public-Key Cryptography. In: Proceedings of the IEEE, Vol. 76 (1988), No. 5, S. 560–577.
http://zoo.cs.yale.edu/classes/cs426/2012/bib/diffie88first.pdf

Diffie, W.; Hellman, M.E.: New Directions in Cryptography. In: IEEE Transactions on Information Theory, 22. Jg. (1978), Nr. 6, S. 644–654.
http://www-ee.stanford.edu/~hellman/publications/24.pdf

EFF – Electronic Frontier Foundation: Third Annual EFF Pioneer Awards. 1994.
https://w2.eff.org/awards/pioneer/1994.php

Elgamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: G.R. Blakley, D. Chaum (Hrsg.): Advances in Cryptology – Proceedings of CRYPTO 84. Reihe „LNCS – Lecture Notes in Computer Science “, Band 196. Berlin u.a.: Springer, 1985, S. 10–18.
http://link.springer.com/chapter/10.1007%2F3-540-39568-7_2#

Ermert, M.: Nach Snowden – Wenig Schlaf für Kryptoforscher. heise Security im Gespräch mit der Kryptologin Tanja Lange. heise Security, 17.09.2014.
http://www.heise.de/security/artikel/Nach-Snowden-Wenig-Schlaf-fuer-Kryptoforscher-2392236.html

Esslinger, B. u.a.: Das CrypTool-Skript – Kryptographie, Mathematik und mehr. Hintergrundmaterial und Zusatzinformationen zum freien E-Learning-Programm CrypTool (mit Code-Beispielen zur Zahlentheorie, geschrieben in Sage). [11]2013.
https://www.cryptool.org/images/ctp/documents/CrypToolScript-de-draft.pdf

Hetland, M.L. (deutsche Übersetzung: Hartmut Pfüller): Instant Python. 5. Oktober 2014.
http://lodda.github.io/instantpython.html

IMDb – Internet Movie Database: Der Beweis, 2005.
http://www.imdb.com/title/tt0377107/

Institut für Telematik: Nachrichten-Verschlüsselung und Digitales Signieren. 2001.
https://www.cryptoportal.org/details.php?PHPSESSID=5150eea8888f45a5fcf578f467dce414&tu_id=91

JCrypTool (JCT):
https://www.cryptool.org/de/jcryptool

Kardel, F.: Die Falltürfunktion als mathematische Grundlage für eine Codierung und Decodierung auf dem Kleincomputer. In: LOG IN, 4. Jg. (1984) – Teil 1: Heft 1, S. 56–62; Teil 2: Heft 2, S. 61–62; Teil 3: Heft 3, S. 62–64.

Karpfinger, Chr.; Kiechle, H.: Kryptologie – Algebraische Methoden und Algorithmen. Reihe „Studium“. Wiesbaden: Vieweg+Teubner, 2010.

Lang, H.W.: Kryptografie – Kryptografische Hashfunktion. 29.05.2015a.
http://www.inf.fh-flensburg.de/lang/krypto/hashfunktion.htm

Lang, H.W.: Kryptografische Protokolle – ElGamal-Verschlüsselung. 29.05.2015b.
http://www.inf.fh-flensburg.de/lang/krypto/protokolle/elgamal.htm

Lautebach, U.: Neee, das hab' ich nie gesagt! Das Chatprotokoll Off-the-Record (OTR). In: LOG IN, 35. Jg. (2015), Heft 181/182, S. 77–84 *(in diesem Heft)*.

Levy, St.: Crypto – How the Code Rebels Beat the Government – Saving Privacy in the Digital Age. New York: Viking Penguin, 2001.

Müller, J.: Einwegfunktionen. In: LOG IN, 31. Jg. (2011), Nr. 169/170, S. 106–111.

Niedermeier, R.; Vogel, J.; Fothe, M.; König, M.: Das Knotenüberdeckungsproblem – Eine Fallstudie zur Didaktik NP-schwerer Probleme. In: LOG IN, 27. Jg. (2007) – Teil 1: Nr. 146/147, S. 53–59; Teil 2: Nr. 148/149, S. 81–89.

Python Software Foundation: The Python Tutorial. 1990 ff.
https://docs.python.org/2/tutorial/

SageMath (Computeralgebrasystem):
http://www.sagemath.org/

Schneier, B.: Attacking Tor – how the NSA targets users' online anonymity. The Guardian, 4. Oktober 2013.
http://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity

Schulz, R.-H.; Witten, H.; Esslinger, B.: Rechnen mit Punkten einer elliptischen Kurve. In: LOG IN, 35. Jg. (2015), Heft 181/182, S. 103–115 *(in diesem Heft)*.

Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. 25.01.1996.
http://arxiv.org/pdf/quant-ph/9508027v2.pdf

SIGSAM: Richard Dimick Jenks Memorial Prize – 2013 Award:
http://www.sigsam.org/awards/jenks/awardees/2013/

Silverman, J.H.; Tate, J.: Rational Points on Elliptic Curves. Reihe „Undergraduate Texts in Mathematics“. New York u.a.: Springer, 1992.

Singh, S.: Geheime Botschaften – Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet. München: Deutscher Taschenbuch Verlag, [10]2011 [Erstausgabe: The Code Book – The Science of Secrecy from Ancient Egypt to Quantum Cryptography. London: Fourth Estate, 1999].

Stein, W.: Elementary Number Theory – Primes, Congruences, and Secrets. 16. November 2011.
http://wstein.org/ent/ent.pdf

Wätjen, D.: Kryptographie – Grundlagen, Algorithmen, Protokolle. Heidelberg; Berlin: Spektrum Akademischer Verlag, 2004.

Wikipedia – Stichwort „Diffie-Hellman-Schlüsselaustausch“:
http://de.wikipedia.org/wiki/Diffie-Hellman-Schlüsselaustausch

Wikipedia – Stichwort „Discrete logarithm records“:
https://en.wikipedia.org/wiki/Discrete_logarithm_records

Wikipedia – Stichwort „Eavesdropping“:
http://en.wikipedia.org/wiki/Eavesdropping

Wikipedia – Stichwort „Elgamal-Signaturverfahren“:
https://de.wikipedia.org/wiki/Elgamal-Signaturverfahren

Wikipedia – Stichwort „Elliptic Curve DSA“:
https://de.wikipedia.org/wiki/Elliptic_Curve_DSA

Wikipedia – Stichwort „Körper (Algebra)“:
https://de.wikipedia.org/wiki/K%C3%B6rper_(Algebra)

Wikipedia – Stichwort „Pohlig-Hellman-Algorithmus“:
https://de.wikipedia.org/wiki/Pohlig-Hellman-Algorithmus

Wikipedia – Stichwort „Safe prime“:
http://en.wikipedia.org/wiki/Safe_prime

Wikipedia – Stichwort „SageMath“:
https://en.wikipedia.org/wiki/SageMath

Wikipedia – Stichwort „Sophie Germain prime“:
http://en.wikipedia.org/wiki/Sophie_Germain_prime

Wikipedia – Stichwort „Strong prime“:
http://en.wikipedia.org/wiki/Strong_prime

Wikipedia – Stichwort „Taher Elgamal“:
https://de.wikipedia.org/wiki/Taher_Elgamal

Wikipedia – Stichwort „Whitfield Diffie“:
https://en.wikipedia.org/wiki/Whitfield_Diffie

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 1: RSA für Einsteiger. In: LOG IN, 26. Jg. (2006a), Heft 140, S. 45–54.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 2: RSA für große Zahlen. In: LOG IN, 26. Jg. (2006b), Heft 143, S. 50–58.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 3: RSA und die elementare Zahlentheorie. In: LOG IN, 28. Jg. (2008), Heft 152, S. 60–70.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 4: Gibt es genügend Primzahlen für RSA? In: LOG IN, 30. Jg. (2010a), Heft 163/164, S. 97–103.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 5: Der Miller-Rabin-Primzahltest oder: Falltüren für RSA mit Primzahlen aus Monte Carlo. In: LOG IN, 30. Jg. (2010b), Heft 166/167, S. 92–106.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 6: Das Faktorisierungsproblem oder: Wie sicher ist RSA? In: LOG IN, 31. Jg. (2011/2012), Heft 172/173, S. 59–69.

Alle Internetquellen wurden zuletzt am 15. September 2015 geprüft und können auch aus dem Service-Bereich des LOG IN Verlags (http://www.log-in-verlag.de/) heruntergeladen werden.

Anzeige

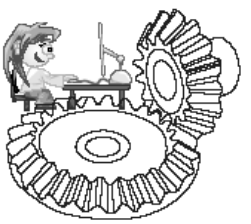
Herausgeber:
Prof. Dr. paed. habil. Norbert Breier
Prof. Dr. paed. habil. Steffen Friedrich
Kerstin Schacht


ISBN 3-89818-624-5 (2. Aufl.)
176 Seiten, vierf.
15,95 Euro

Lehrermaterial
ISBN 3-89818-613-X
15,95 Euro

**Technik und Computer**

*Das neue Lehrbuch für das Fach „Technik und Computer“ Kl. 5 und 6, optimiert für die Lehrpläne Sachsen, Mittelschule und Gymnasium*



- **Von der Idee zum Produkt**
  (Fertigungsunterlagen; Werkstoffe; Prüfen und Messen; Fertigungsverfahren und ihr Einsatz – Urformen, Umformen, Trennen, Fügen, Beschichten; Herstellen eines einfachen Werkstücks)
- **Wir untersuchen mechanische Objekte**
  (Maschinen – Nutzen, Bauteile, Bewegungsübertragung; Lösen von Fertigungsaufgaben – Bau eines Modells, Untersuchen eines technischen Objekts)
- **Wie war es gestern – wie wird es morgen?**
  (Vom Faustkeil zum Roboter, Handwerksberufe gestern und heute, Vom Rad zum Düsenjet, Von der Nutzung des Feuers zur Windkraftanlage)
- **Der Computer – ein Arbeitsgerät mit Zukunft**
  (Auf den ersten Blick, Hardware und Software, Eingabe – Verarbeitung – Ausgabe, Speichern von Daten und Programmen, Umgang mit Computern, Im Gehirn eines Computers, Betriebssystem und Dateiverwaltung)
- **Textverarbeitung – vom Buchstaben zum Buch**
  (Schreiben als Weitergabe von Information; Zeichen, Wörter, Zeilen und Absätze; Gestaltung eines Textes; Rechtschreibhilfe; Textkorrektur; Arbeit mit Textverarbeitungsprogrammen; Schreibregeln; Buchstaben, Ziffern, Sonderzeichen; Erstellung einer Tabelle, Erstellung einer Dokumentation)
- **Kommunikation: gestern – heute – morgen**
  (Kommunikation – was ist das eigentlich? Fackelzeichen, Zeigertelegrafen, Morsecode, Telefonieren und mehr, Das Internet nutzen, Im Internet suchen)

DUDEN PAETEC
SCHULBUCHVERLAG