Vorlesung Kryptologie

Enno Ohlebusch

Sommersemester 2024

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Organisatorisches

- Vorlesung : Montag, 16:15–17.45 Uhr, H20
- Vorlesung : Mittwoch, 14:15–15.45 Uhr, H21
- Übung: (zweiwöchentlich), je nach Bedarf
- Veranstaltung im Bachelor- und Masterstudiengang
- 4 SWS (3V+1Ü), 6LP
- kein Notenbonus
- Prüfung: Klausur

Literatur

- Johannes Buchmann, Einführung in die Kryptographie, 4. erweiterte Auflage, Springer-Verlag, 2008.
- Joachim von zur Gathen, Cryptoschool, Springer-Verlag, 2015.
- Douglas R. Stinson, Cryptography: Theory and Practice, Second Edition, Chapman & Hall/CRC, 2002.
- Dietmar Wätjen, Kryptographie: Grundlagen, Algorithmen, Protokolle, 3.Auflage, Springer Vieweg, 2018.
- Hans Delfs und Helmut Knebl, Introduction to Cryptography, Springer-Verlag, 2015.

Begriffe: Kryptologie

Der Oberbegriff Kryptologie umfasst

- Kryptographie (Methoden der Ver- und Entschlüsselung)
- Kryptoanalyse (wie und mit welchem Aufwand man Krypto-Systeme ggf. knacken kann)

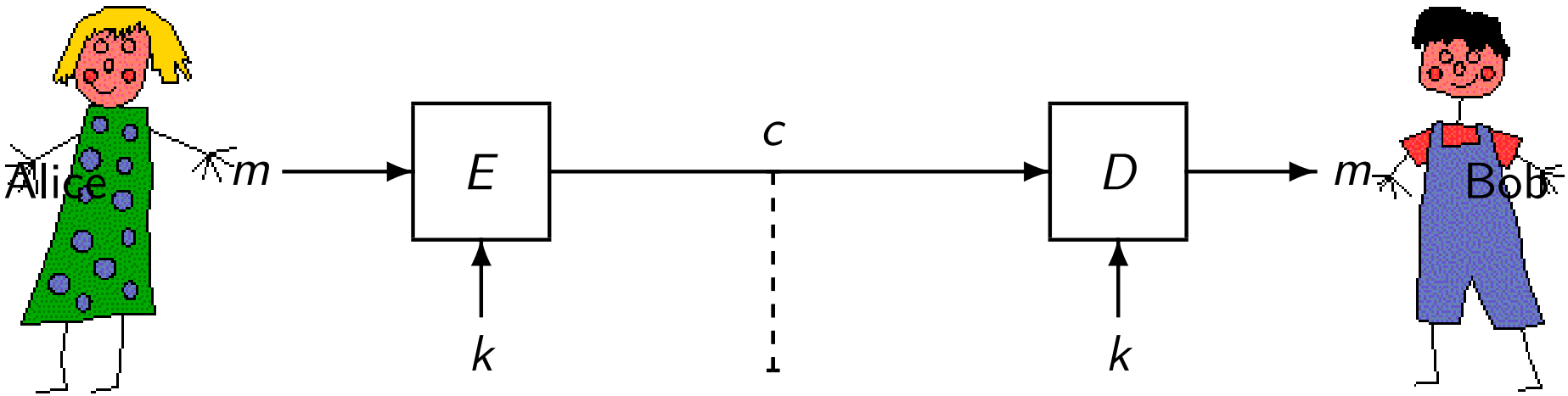
Begriffe: Kryptographie

- Ziel der Kryptographie ist es, eine geheime Botschaft über einen unsicheren Kanal zu verschicken, sodass nur der beabsichtigte Empfänger die Botschaft lesen kann.
- Dieses Problem beschäftigt Menschen schon seit Jahrtausenden.
- Ein berühmtes Kryptoverfahren wurde von Julius Cäsar benutzt, um mit seinen Offizieren zu kommunizieren (Cäsar-Chiffre).
- Klassische Verfahren basieren auf der Annahme, dass die Parteien, die Informationen austauschen möchten, sich vorher auf einen geheimen Schlüssel geeinigt haben.
- Solche Verfahren nennt man symmetrische Verfahren, weil der gleiche Schlüssel sowohl zum Ver- als zum Entschlüsseln benutzt wird.

Begriffe: Klassische Kryptographie

- Der Sender hat Nachricht m, einen Verschlüsselungsalgorithmus E, der m und einen geheimen Schlüssel k als Eingabe nimmt und die Chiffre $c = E(m, k)$ berechnet.
- Diese Chiffre wird über einen potenziell abhörbaren Kanal geschickt.
- Der Empfänger verwendet denselben Schlüssel k für den Dechiffrieralgorithmus D und berechnet $D(c, k) = m$.
- Wie der Schlüssel k zuvor zwischen Sender und Empfänger ausgetauscht/vereinbart wurde, wird als gegeben angenommen.
- Die Algorithmen zur Berechnung von E bzw. D sind im Allgemeinen deterministisch und sehr effizient.
- Gängige Bezeichnung für Sender, Empfänger und Abhörer ist Alice, Bob und Eve (für eavesdropper)

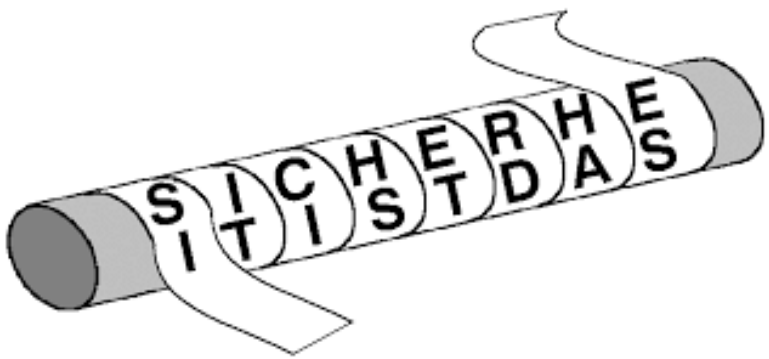
Begriffe: Klassische Kryptographie

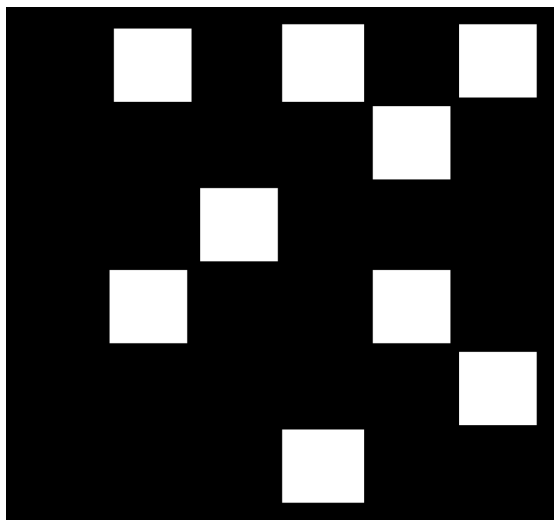


Begriffe: Transpositions- und Substitutionschiffren

Bei Transpositionschiffren bleiben die Klartextbuchstaben unverändert, nur ihre Position in der Chiffre wird verändert.

- z.B. Skytale im antiken Griechenland: Leder- oder Papierband über Stab wickeln, seitlich beschriften und wieder abwickeln
- z.B. Löcherschablonen: die Buchstaben in die Löcher eintragen, danach die Schablone drehen und weiter beschriften





Bei einer Substitutionschiffre werden die Klartextbuchstaben durch andere ersetzt.

Begriffe: Monoalphabetisch und polyalphabetisch

Monoalphabetisch:

- Einzelbuchstaben werden durch andere Buchstaben (oder irgendwelche Symbole) ersetzt
- derselbe Buchstabe erfährt jedes Mal dieselbe Ersetzung (in der Antike und im Mittelalter üblich, z.B. Cäsar-Chiffre)

Polyalphabetisch:

- derselbe Buchstabe wird nicht jedes Mal auf dasselbe Zeichen abgebildet
- die Ersetzung ändert sich nach einem bestimmten Schema jedes Mal

Begriffe: Blockchiffren und Stromchiffren

Blockchiffren und Stromchiffren sind polyalphabetische Chiffrierverfahren.

- Bei Blockchiffren werden jeweils Blöcke von Klartext-buchstaben (mit einer festen Blocklänge, z.B. 5 bis ca. 100) in Chiffreblöcke (meist derselben Länge) abgebildet.
- Bei Stromchiffren besitzt der Chiffrieralgorithmus einen inneren Zustand.
- Der innere Zustand ändert sich mit jedem verschlüsselten Klartextbuchstaben.
- Die Chiffrierung des nächsten Klartextbuchstaben hängt auch vom momentanen inneren Zustand ab.

Begriffe: Lawineneffekt

Eine wünschenswerte Eigenschaft guter Krypto-Systeme ist der *Lawineneffekt*:

- das Ändern nur eines Klartext-Buchstabens oder -Bits sollte eine große Änderung des Chiffretextes hervorrufen (so als ob die Chiffre-Buchstaben oder Bits gegenüber der vorherigen Situation zufällig und unabhängig neu festgelegt würden).

Der Lawineneffekt ist sicherlich nicht vorhanden

- bei einer monoalphabetischen Substitutionschiffre
- bei einer Transpositionschiffre

Begriffe: Kerckhoff-Prinzip

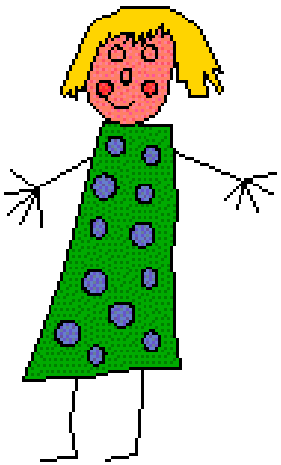
Das Kerckhoff-Prinzip besagt, dass die verwendeten Chiffrier- und Dechiffrier-Algorithmen E und D als bekannt vorausgesetzt werden (heutzutage Standard).

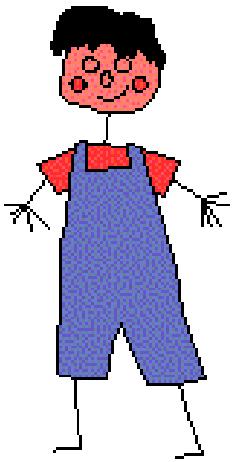
Bei vielen historischen Verfahren wurde dagegen das Verfahren an sich geheim gehalten, das dann auch ohne weiteren Schlüssel verwendet wurde.

Begriffe: Asymmetrische Kryptographie

- Der Empfänger Bob erzeugt in der Initialisierungsphase zwei Schlüssel:
  - einen öffentlichen Schlüssel k, den frei zugänglich ist (z.B. auf Bobs Homepage),
  - und einen geheimen Schlüssel k', den nur der Empfänger kennt.
- Die Schlüssel k, k' sind solcherart invers zueinander, dass gilt: $E(m, k) = c$ und $D(c, k') = m$.
- Die Ver- und Entschlüsselungsalgorithmen E und D werden als öffentlich bekannt angenommen (Kerckhoff-Prinzip).
- Für einen Angreifer soll es praktisch unmöglich sein, aus k den Schlüssel k' zu berechnen. Daher spricht man hier von asymmetrisch.

Asymmetrische Kryptographie: Beispiel RSA





Init

(n, e)

(n, e) d

m $\quad c = m^e \bmod n \quad$ $c^d \bmod n = m$

Begriffe: Moderne Kryptographie

Das Entscheidende bei moderner Kryptographie ist, dass

- jedwede Kommunikation zwischen Alice und Bob über den abhörbaren Kanal zu erfolgen hat und
- keinerlei vorheriger Schlüsselaustausch (wie bei der klassischen Kryptographie) stattfindet.

D.h. moderne Kryptographie muss nicht notwendigerweise immer durch die oben beschriebene asymmetrische Schlüsselsituation charakterisiert sein.

Begriffe: Algorithmen, Protokolle und Runden

- Verschiedene Algorithmen müssen beschrieben werden, z.B. zur Berechnung von E auf seiten von Alice, und zur Berechnung von D auf seiten von Bob.
- Solche Algorithmen können deterministisch oder auch probabilistisch sein.
- In der modernen Kryptographie müssen meist mehrere Nachrichtaustausche erfolgen: Alice sendet an Bob, Bob wieder an Alice, usw.
- Wir sprechen dann von einem Protokoll, das in verschiedenen Runden stattfindet.
- Jede Runde erfordert ggf. seinen eigenen Algorithmus.

Begriffe: Hybride Kryptosysteme

- Die Verschlüsselungs- und Entschlüsselungsalgorithmen der modernen Kryptographie erfordern deutlich mehr Aufwand als bei der klassischen Kryptographie ($O(n^3)$ versus $O(n)$).
- Deshalb werden die Methoden der modernen Kryptographie oft nur dazu verwendet, einen Schlüssel, und nicht die ganze Nachricht, auszutauschen/zu vereinbaren.
- Diese eigentliche Nachricht wird nun mit einem als sicher erachteten klassischen Verfahren übertragen, wobei dabei der zuvor ausgetauschte Schlüssel verwendet wird.
- Zum Beispiel ist das Diffie-Hellman-Verfahren immer ein solches hybrides Verfahren.

Begriffe: Angriffsszenarien

- Cyphertext-only: Angreifer kennt c und soll m (evtl. auch k) ermitteln.
- Known-Plaintext: Angreifer kennt ein (oder einige) (m, c) mit $c = E(m, k)$ und soll k ermitteln.
- Chosen-Plaintext: Angreifer kann m frei wählen und kann $E(m, k) = c$ berechnen. Damit soll k ermittelt werden.
- Chosen-Cyphertext: Angreifer kann c frei wählen und dazu $m = D(c, k)$ berechnen. Nun soll er damit k ermitteln.
- Man-in-the-middle (bei moderner Kryptographie): Sofern dies technisch möglich ist, könnte der Angreifer Eve sich aktiv zwischen Alice und Bob schalten. Für Alice gibt sich Eve als Bob aus; für Bob gibt sich Eve als Alice aus. Nicht nur kann Eve die Nachrichten von Alice und Bob lesen, sondern auch verändern.

Begriffe: Sicherheitsniveau

- Die Sicherheit eines heutigen Krypto-Systems soll nur von der Länge des Schlüssels abhängen und der Art und Weise wie Nachricht und Schlüssel miteinander verrechnet werden.
- Im Idealfall besteht die einzige Möglichkeit, ein System zu knacken, darin, alle potenziellen Schlüssel der verwendeten Schlüssellänge (z.B. k Bits) durchzuprobieren.
- Das wäre ein Aufwand von 2^k.
- Wenn das so ist, so gilt eine Schlüssellänge von $k \geq 80$ als sicher. Das Sicherheitsniveau ist dann k.

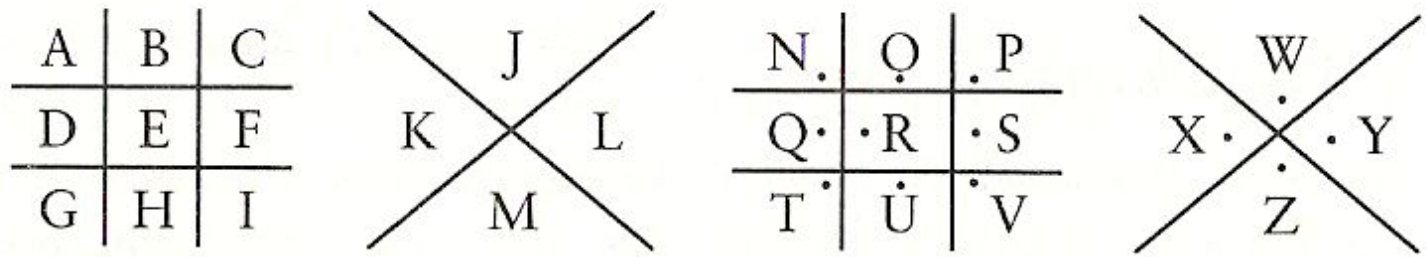
Begriffe: Sicherheitsniveau

- Wenn es einen Algorithmus zum Knacken des Krypto-Systems gibt, welcher bei einem Schlüssel der Länge k einen Rechenaufwand von $t(k) < 2^k$ hat, so beträgt das Sicherheitsniveau $\log_2 t(k)$.
- Um das Mindestsicherheitsniveau von 80 zu erreichen, muss die verwendete Schlüssellänge k so gewählt werden, dass $t(k) \geq 2^{80}$.
- Die Sicherheit des Kryptosystems RSA beruht z.B. darauf, dass es nur mit sehr großem (also exponentiellem) Aufwand möglich ist, eine k-Bitzahl in ihre Primfaktoren zu zerlegen (für das RSA-Modul n gilt $k = \lceil \log_2 n \rceil$).
- Wenn wir (einigermaßen realistisch) davon ausgehen, dass es Faktorisierungsalgorithmen mit Laufzeit $2^{k/10}$ gibt, so benötigt man, um das Sicherheitsniveau ≥ 80 zu erreichen, eine Schlüssellänge von ≥ 800 Bit.

Historische Chiffren: Cäsar-Chiffre

- Ursprünglich wurde jeder Buchstabe um 3 Positionen im Alphabet verschoben.
- Sei $\{A, B, C, \ldots, Z\} = \{0, 1, \ldots, 25\}$. Dann ist (ohne weiteren Schlüssel) $E(x) = (x + 3) \bmod 26$.
- Das Verschlüsselungsverfahren E wurde dabei geheim gehalten.
- Man kann dies so verallgemeinern, dass es 26 Schlüssel $k \in \{0, 1, 2, \ldots, 25\}$ gibt so, dass $E(x, k) = (x + k) \bmod 26$.
- Die Umkehrfunktion ist $D(c, k) = (c - k) \bmod 26$.
- Die Cäsar-Chiffrierung mit Schlüssel $k = 13$ ist selbst-invers: Ver- und Entschlüsselung verwendet dieselbe Funktion, $E(E(x)) = x$.

Historische Chiffren: Freimaurer-Chiffre



Zum Beispiel ist

nichts anderes als das Wort KRYPTOGRAPHIE.

Die Operationen div und mod

Seien $a \in \mathbb{Z}$ und $b \in \mathbb{N}$. Dann existiert eine eindeutig bestimmte Darstellung

$$a = x \cdot b + r,$$

wobei $x \in \mathbb{Z}$ und $r \in \{0, 1, 2, \ldots, n-1\}$.
Die Operationen div und mod sind definiert durch

$$a \text{ div } b = x$$

$$a \text{ mod } b = r$$

Es gilt also

$$a = x \cdot b + r = (a \text{ div } b) \cdot b + (a \text{ mod } b)$$

Beispiel: $a = -60$ und $b = 26$.

Affine Chiffre

- Der Schlüssel besteht aus 2 Komponenten $k = (a, b)$ und man rechnet $E(x, (a, b)) = (a \cdot x + b) \bmod 26$.
- Damit Injektivität gewährleistet ist, muss $ggT(a, 26) = 1$ gelten.
- Das heißt, für a kommen nur die 12 Zahlen $1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25$ in Frage.
- Diese Menge der zu n teilerfremden Zahlen zwischen 1 und $n - 1$ bezeichnen wir mit $\mathbb{Z}_n^*$ (später mehr dazu).
- Es gibt $12 \cdot 26 = 312$ viele mögliche Schlüssel $k \in \mathbb{Z}_{26}^* \times \mathbb{Z}_{26}$.
- Jedes $a \in \mathbb{Z}_n^*$ besitzt genau ein multiplikatives Inverses in $\mathbb{Z}_n^*$, das wir mit a' bezeichnen; d.h. $a \cdot a' \equiv 1 \pmod{n}$.
- Im Beispiel sind paarweise invers zueinander: $(1, 1)$, $(3, 9)$, $(5, 21)$, $(7, 15)$, $(11, 19)$, $(17, 23)$, $(25, 25)$.

Affine Chiffre

Zahlenbeispiel: Sei $c = (7x + 4) \bmod 26$. Die Zeichen $(0, 1, \ldots, 25)$ werden also abgebildet auf

$$(4, 11, 18, 25, 6, 13, 20, 1, 8, 15, 22, 3, 10, 17, 24, 5, 12, 19, 0, 7, 14, 21, 2, 9, 16, 23)$$

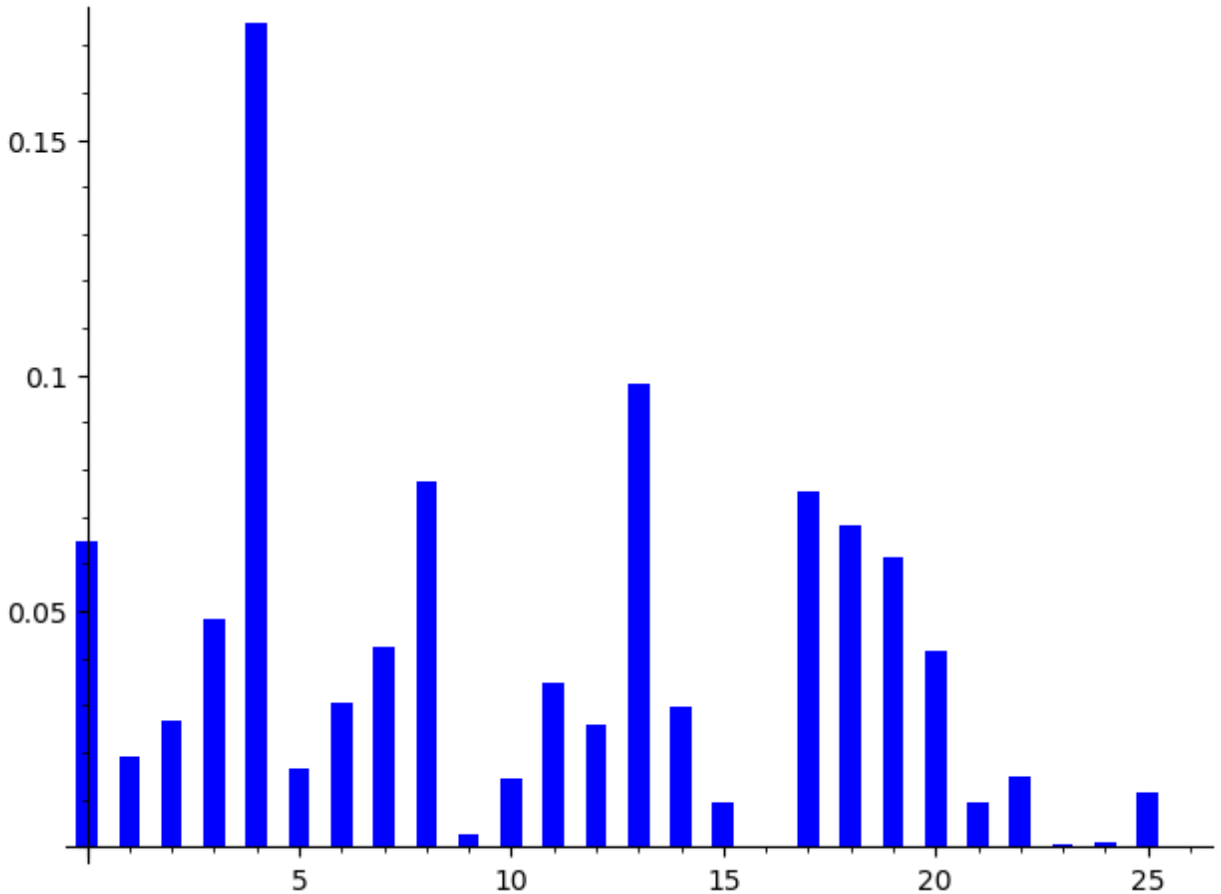
Dechiffrierung:

- Für den Chiffre-Buchstaben c gilt: $c = (7x + 4) \bmod 26$.
- Indem wir mit 15 (dem multiplikativen Inversen von 7) multiplizieren, erhalten wir: $15c = (x + 60) \bmod 26$.
- Es gilt : $x = (15c - 60) \bmod 26$.
- Also: $x = (15c + 18) \bmod 26$.
- Die Dechiffrierung kann also auch als eine affine Chiffrierung aufgefasst werden

Allgemeine monoalphabetische Chiffrierung

- Jeder der 26 Buchstaben kann auf einen beliebigen anderen Buchstaben abgebildet werden (in bijektiver Abb. $\Sigma \to \Sigma$).
- Die Menge der Schlüssel ist somit die Menge aller Permutationen auf 26 Elementen und deren Anzahl ist $26! \approx 2^{88}$.
- Obwohl die Anzahl der Schlüssel anscheinend ausreichend groß ist, liegt kein Sicherheitsniveau von 88 Bit vor.
- Ab einer Textlänge von 30 bis 50 Buchstaben lässt sich eine derartige Chiffre (ohne den Schlüssel zu kennen, also cyphertext-only) relativ leicht entschlüsseln.
- Die Entschlüsselung basiert auf den typischen Buchstabenhäufigkeiten der deutschen Sprache, die sich bei einer monoalphabetischen Chiffrierung ebenso in der Chiffre widerspiegeln.

Buchstabenhäufigkeiten im Deutschen



A	6.47%	N	9.82%
B	1.93%	O	2.98%
C	2.68%	P	0.96%
D	4.83%	Q	0.02%
E	17.48%	R	7.54%
F	1.65%	S	6.83%
G	3.06%	T	6.13%
H	4.25%	U	4.17%
I	7.73%	V	0.94%
J	0.27%	W	1.48%
K	1.46%	X	0.04%
L	3.49%	Y	0.08%
M	2.58%	Z	1.14%

Homophone Chiffrierung

- Jedem Klartextbuchstaben wird eine Menge von möglichen Chiffrezeichen zugeordnet, deren Anzahl proportional zur Auftretenshäufigkeit des Klartextbuchstabens im Deutschen ist.
- Die gesamte Tabelle stellt den geheimen Schlüssel dar.
- Beim Chiffrieren wird ein Zeichen aus der betreffenden Menge jeweils per Zufall ausgewählt.
- Dadurch wird erreicht, dass die Auftretenswahrscheinlichkeit der Chiffrezeichen nahezu einer Gleichverteilung entspricht.
- Angreifbar ist ein solches Chiffriersystem immer noch über die Bigramm- und Trigramm-Häufigkeiten.

Homophone Chiffrierung

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
09	78	48	13	45	25	39	65	83	51	84	22	58	71	95	29	35	40	76	49	61	89	28	21	52	66
12	92	81	41	79	23	50	68	88			27	59	91	94			42	86	69	63					
33			62	14		56	32	93			18		00				77	96	75	34					
47			01	16			70	15					05				80	17	85	60					
53			03	24			73	04					07				11	20	97						
67				44				26					54				19	30	08						
				46				37					72				36	43							
				55				58					90												
				57									99												
				64									38												
				74																					
				82																					
				87																					
				98																					
				10																					
				31																					
				06																					

Beispiel für eine homophone Verschlüsselung.

Playfair-Chiffre

- Eine Playfair-Chiffre verschlüsselt aufeinander folgende Paare von Buchstaben (Bigramme) wiederum zu Bigrammen.
- Sie ist also eine Blockchiffre mit Blocklänge 2.
- Allerdings sind Paare von gleichen Buchstaben nicht vorgesehen. (Notfalls das Wort nur mit einem statt mit einem Doppelbuchstaben schreiben, oder einen Füllbuchstaben wie X einfügen.)
- Man verwendet zum Chiffrieren und zum Dechiffrieren eine quadratische 5×5 Tabelle.
- Dazu werden die Buchstaben I und J miteinander identifiziert, so dass man ein Alphabet mit nur 25 Buchstaben verwendet.
- Das geheime Schlüsselwort (der Länge 5) wird in die erste Zeile der Tabelle geschrieben; es sollte keinen doppelt vorkommenden Buchstaben enthalten.

Playfair-Chiffre mit Schlüsselwort DEATH

Man lässt alle bisher nicht vorgekommenen Buchstaben nachfolgen und füllt die Tabelle vollständig aus.

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

Verschlüsseln zweier Buchstaben mit Schlüsselwort DEATH

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

- Wenn die beiden Buchstaben in derselben Zeile liegen, so nimmt man die in derselben Zeile nachfolgenden zwei Buchstaben (mit wrap-around). Beispiel: aus LO wird MK.
- Wenn die beiden Buchstaben in derselben Spalte liegen, so nimmt man die in derselben Spalte nachfolgenden zwei Buchstaben (mit wrap-around). Beispiel: aus FM wird MR.
- Wenn die beiden Buchstaben die diagonal gegenüber liegenden Ecken eines Rechtecks bilden, so nimmt man (im Uhrzeigersinn) die zwei Buchstaben an den anderen beiden Ecken. Beispiel: aus UF wird RI; aus AP wird RD.

Vigenère-Chiffre

- Die Vigenère-Chiffre aus dem 16. Jahrhundert galt über Jahrhunderte als sicher, bis sie ca. 1850 geknackt wurde.
- Man wählt ein Schlüsselwort, etwa DEATH, und schreibt dieses fortlaufend wiederholt unter den Klartext.
- Dann werden die Klartext- und die Schlüsselwortbuchstaben (ohne Übertrag) modulo 26 addiert.
- Das heißt, man identifiziert die Buchstaben $A, B, \ldots, Z$ mit den Zahlen $0, 1, \ldots, 25$.
- Das Schlüsselwort DEATH entspricht somit den Zahlen 3, 4, 0, 19, 7.

Vigenère-Chiffre: Beispiel

```
  ANKOM MEFRE ITAGD ENDRE IZEHN TEN
+ DEATH DEATH DEATH DEATH DEATH DEA
  ----------------------------------
= DRKHT PIFKL LXAZK HRDKL LDEAU WIN
```

- In der Chiffre findet sich an 2 Stellen das Trigramm KLL, und zwar im Abstand von 10 Buchstaben (was zweimal der Schlüsselwortlänge entspricht).
- Bei Trigrammen sollte ein solches Doppelvorkommen eigentlich sehr unwahrscheinlich sein.
- Der Grund für das Doppelvorkommen von KLL liegt daran, dass im Klartext an der betreffenden Stelle das im Deutschen sehr häufige Trigramm REI vorkommt, und darüber hinaus mit derselben Schlüsselwort-Buchstabenfolge, nämlich THD, chiffriert wurde.

Knacken der Vigenère-Chiffre

- Die Kasiski-Methode zum Bestimmen der Schlüsselwortlänge geht so vor: finde mehrere solcher Mehrfachvorkommen von Trigrammen in der Chiffre und bestimme deren Abstände.
- Wenn man z.B. die Abstände n_1, n_2, n_3 kennt, so ist $\ell = ggT(n_1, n_2, n_3)$ sehr wahrscheinlich die verwendete Schlüsselwortlänge.
- Wenn man die Schlüsselwortlänge ℓ richtig bestimmt hat, geht es nur noch darum, die Cäsar-Chiffrierungen, die jeweils im Abstand ℓ mit demselben Schlüsselbuchstaben stattfinden, aufzudecken.
- Dies geht mit einer Analyse der Buchstabenhäufigkeiten.
- Das genaue Vorgehen wird später besprochen (Methode zum Knacken der Vigenère-Chiffre).

ENIGMA

- ENIGMA ist ein von den Deutschen im 2. Weltkrieg verwendetes polyalphabetisches Chiffriergerät, das auf dem Prinzip einer Stromchiffre funktioniert.
- Es besteht aus einem Tastenfeld und einem Feld mit Lämpchen.
- Bei Betätigen einer Taste zeigt eine Lampe dann den entsprechenden Chiffre-Buchstaben an.
- Nach demselben Prinzip funktioniert auch das Dechiffrieren.

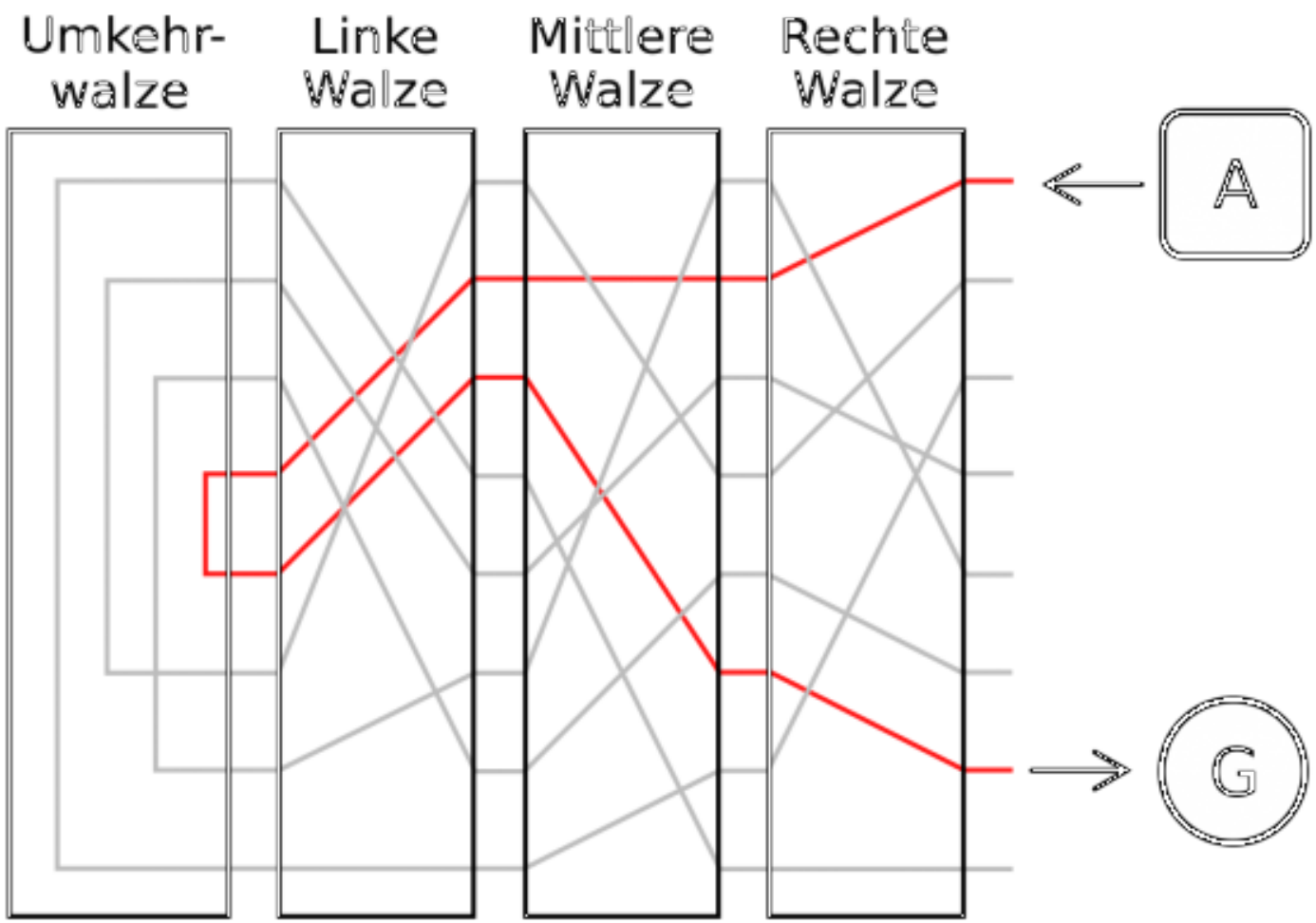
ENIGMA

ENIGMA

Je nach Ausführung besitzt eine ENIGMA drei oder vier Walzen, die auf jeder Seite mit 26 Kontakten versehen sind und im Inneren wird gemäß einer Permutation der 26 Buchstaben die Vorder- und die Rückseite miteinander verdrahtet.

- Mit jedem Tastendruck dreht sich die rechte Walze um eine Position weiter; nach einer ganzen Drehung wird dann auch die nächste Walze um 1 weitergedreht, usw.
- Der Stromverlauf geht von der gedrückten Taste durch die Walzen.
- Am linken Ende wird wieder mit einer Permutation das elektrische Signal von links nach rechts durch die Walzen geschickt, um am Ende ein entsprechendes Lämpchen aufleuchten zu lassen.
- Gegebenenfalls geht der Stromverlauf auch noch durch ein Steckfeld.

ENIGMA



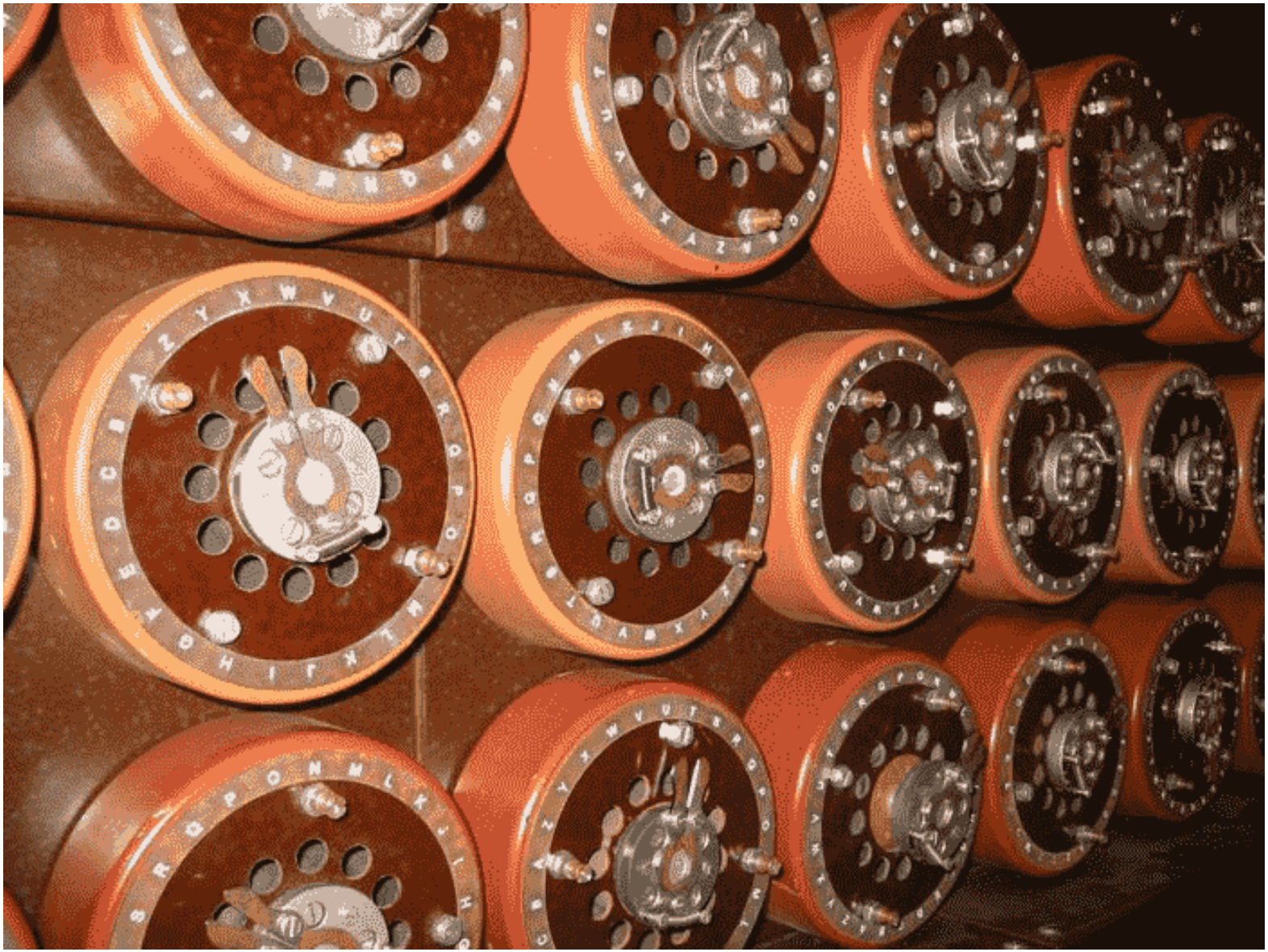
ENIGMA

Sender und Empfänger müssen dieselbe Ausgangsstellung haben; dies ist der Schlüssel und besteht aus folgenden Informationen:

- Welche der zur Verfügung stehenden Walzen und in welcher Reihenfolge diese einzusetzen sind.
- Welches die durch Drehung einzustellende Anfangspositionen der Walzen sind.
- Welche Steckverbindungen auf dem Steckfeld einzustellen sind.

Vom Mathematiker Alan Turing wurde der ENIGMA-Code geknackt. Hierzu verwendete er kreisförmig rotierende Relais-Drehscheiben, Turing-Bombe genannt, die die möglichen Walzenstellungen simulierten.

Turing-Bombe



Koinzidenzindex

- Sei $p = (p_1, p_2, \ldots, p_k)$ eine Wahrscheinlichkeitsverteilung wobei $p_i \in [0, 1]$ und $\sum_{i=1}^{k} p_i = 1$.
- Dann ist der Koinzidenzindex (index of coincidence) von p definiert als

$$IC(p) = \sum_{i=1}^{k} (p_i)^2$$

- Äquivalente Formulierung: Wenn X und Y zwei unabhängige, gemäß p verteilte Zufallsvariablen sind, so ist

$$IC(p) = P(X = Y)$$

- Da man $X = Y$ im Zusammenhang mit Hashing auch als eine Kollision bezeichnet, spricht man statt Koinzidenzindex auch von Kollisionswahrscheinlichkeit.

Koinzidenzindex

- Der kleinstmögliche Wert von IC ist $1/k$. Dieser tritt genau bei einer Gleichverteilung, also alle $p_i = 1/k$, auf.
- Der größtmögliche Wert ist 1. Dieser tritt nur bei einer deterministischen Verteilung auf: genau ein $p_i = 1$, alle anderen Wahrscheinlichkeiten sind 0.
- Nimmt die Zufallsvariable die 26 Buchstaben des Alphabets als mögliche Werte an, so gilt also bei Gleichverteilung $IC = 1/26 = 3.85\,\%$.
- Legt man die Buchstaben-Wahrscheinlichkeiten in deutschen Texten zugrunde, so ist $IC \approx 7.62\,\%$, also etwa doppelt so hoch.
- Im Englischen liegt der Wert etwas niedriger, bei knapp $7\,\%$.

Schätzung des Koinzidenzindex

- Nun sei ein Text gegeben.
- Wir nehmen an, dass die Buchstaben unabhängig und gemäß einer unbekannten Wahrscheinlichkeitsverteilung p gezogen wurden.
- Es soll $IC(p)$ geschätzt werden.
- Dies ist eine typische Aufgabe aus der Statistik. (Gegeben eine Stichprobe, schätze einen unbekannten Parameter der zugrunde liegenden Wahrscheinlichkeitsverteilung.)

Schätzung des Koinzidenzindex

Eine (erwartungstreue) Schätzung erhält man wie folgt:

$$\widetilde{IC} = \sum_a \frac{n(a)}{m} \cdot \frac{n(a)-1}{m-1} = \frac{1}{m \cdot (m-1)} \cdot \sum_a n(a) \cdot (n(a)-1)$$

- Hierbei durchläuft a alle Buchstaben, die in dem Text vorkommen.
- $n(a)$ ist die Anzahl des Vorkommens von a im Text.
- m ist die Textlänge.
- Interpretation: Mit Wahrscheinlichkeit $n(a)/m$ erhält man beim ersten Ziehen den Buchstaben a; beim zweiten Ziehen (ohne Zurücklegen) erhält man wieder den Buchstaben a mit Wahrscheinlichkeit $(n(a)-1)/(m-1)$.

Schätzung des Koinzidenzindex

$$\widetilde{IC} = \sum_a \frac{n(a)}{m} \cdot \frac{n(a)-1}{m-1} = \frac{1}{m \cdot (m-1)} \cdot \sum_a n(a) \cdot (n(a)-1)$$

- Man beachte, dass man nur solche Buchstaben a in der Summe zu berücksichtigen braucht, die im Text mindestens 2-mal vorkommen, die anderen ergeben in der Summe keinen Beitrag.
- Man beachte ferner, dass man die Größe des zugrunde liegenden Alphabets nicht zu kennen braucht.

Schätzung des Koinzidenzindex: Beispiel

Betrachte den Text KOINZIDENZINDEX.

- Hier kommen I und N 3-mal sowie D, Z und E je 2-mal vor.
- Die Textlänge ist $m = 15$.
- Also ergibt sich $\widetilde{IC} = \frac{3 \cdot 2 + 3 \cdot 2 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1}{15 \cdot 14} = 8.6\,\%$.

Beachte:

- Bei einem monoalphabetisch verschlüsselten deutschen Text wird sich wie bei einem unverschlüsselten ein Koinzidenzindex von ca. 7.62 % ergeben.
- Bei einem polyalphabetisch verschlüsselten Text wird der Koinzidenzindex deutlich niedriger sein, sich also in Richtung $1/26 = 3.85$ % bewegen.

Methode zum Knacken von Vigenère

Um die Schlüsselwortlänge ℓ zu bestimmen, probieren wir alle möglichen Werte $\ell = 1, 2, 3, \ldots$ durch und entscheiden uns für den wahrscheinlichsten.

- Der Text sei $c_1 c_2 \ldots c_m$.
- Für einen gegebenen ℓ-Wert definieren wir $s_i = c_i\, c_{i+\ell}\, c_{i+2\ell}\, c_{i+3\ell} \ldots$ für $i = 1, \ldots, \ell$.
- Für alle Texte s_i berechnen wir den Koinzidenzindex.
- Falls ℓ die gesuchte Schlüsselwortlänge ist, so werden die Koinzidenzindizes bei ca. 7 Prozent oder darüber liegen, bei inkorrekten ℓ-Werten deutlich niedriger.

Methode zum Knacken von Vigenère: Beispiel

Der Klartext sei ANKOMMEFREITAGDENDREIZEHNTEN und das Schlüsselwort (der Länge 4) sei BADE:

```
  ANKOMMEFREITAGDENDREIZEHNTEN
+ BADEBADEBADEBADEBADEBADEBADE
  ----------------------------
= BNNSNMHTSELXBGGIODUIJZHLOTHR
```

Wir untersuchen die mögliche Schlüsselwortlänge $\ell = 3$ und betrachten die Texte s_1, s_2, s_3:

```
B S H E B I U Z O R
 N N T L G O I H T
  N M S X G D J L H
```

In s_1 kommt B zweimal vor. Daher ergibt sich der geschätzte IC-Wert: $(2 \cdot 1)/(10 \cdot 9) = 0.022$.
In s_2 kommt sowohl N als auch T zweimal vor. Dies ergibt $(2 \cdot 1 + 2 \cdot 1)/(9 \cdot 8) = 0.055$. Die dritte Zeile ergibt den Wert 0.
Im Mittel haben wir den Wert 0.026, also 2.6 %.

Methode zum Knacken von Vigenère: Beispiel

```
  ANKOMMEFREITAGDENDREIZEHNTEN
+ BADEBADEBADEBADEBADEBADEBADE
  ----------------------------
= BNNSNMHTSELXBGGIODUIJZHLOTHR
```

Wir untersuchen die mögliche Schlüsselwortlänge $\ell = 4$ und betrachten die Texte s_1, s_2, s_3, s_4:

```
B N S B O J O
 N M E G D Z T
  N H L G U H H
   S T X I I L R
```

Die IC-Schätzungen sind: erste Zeile: 0.095, zweite Zeile: 0, dritte Zeile: 0.143, vierte Zeile: 0.047; im Mittel: 0.0713, also 7.13 %.

Die entsprechende Rechnung für $\ell = 5$ ergibt den mittleren IC-Wert von 0.0332. Daher ist sehr stark (und korrekt) zu vermuten, dass die Schlüsselwortlänge 4 ist.

Methode zum Knacken von Vigenère

Sobald man nun den korrekten ℓ-Wert bestimmt hat, müssen die ℓ-vielen Cäsar-Verschlüsselungen gefunden werden, die den Texten $s_1, \ldots, s_\ell$ zugrunde liegen.

- Man durchläuft die 26 möglichen Cäsar-Entschlüsselungen (Buchstaben-Shifts).
- Man muss denjenigen Shift für s_i finden, sodass die Ähnlichkeit zwischen den relativen Häufigkeiten der geshifteten Buchstaben mit den Häufigkeiten der deutschen Sprache am größten werden.
- Das heißt, man muss zwei Wahrscheinlichkeits- (oder Häufigkeits-) Verteilungen $p = (p_1, p_2, \ldots, p_n)$ und $q = (q_1, q_2, \ldots, q_n)$ auf Ähnlichkeit miteinander vergleichen.

Methode zum Knacken von Vigenère

Hier gibt es mehrere Möglichkeiten:

- die Summe der absoluten Differenzen $\sum_{i=1}^{n} |p_i - q_i|$
- die Summe der quadratischen Differenzen $\sum_{i=1}^{n} (p_i - q_i)^2$

Diese Werte sollten minimal werden.

Man kann auch den paarweisen Koinzidenzindex $\sum_{i=1}^{n} p_i \cdot q_i$ bestimmen. Bei großer Ähnlichkeit der Verteilungen wird dieser Wert maximal.

Siehe zum Beispiel oben (bei $\ell = 4$):

- In der dritten Zeile kommt 3-mal H vor.
- Es ist sehr plausibel, dass der Chiffrebuchstabe H dem Klartextbuchstaben E entspricht.
- Was in diesem Fall auch korrekt ist, denn der dritte Schlüsselbuchstabe ist D, was einer Verschiebung um 3 Buchstaben (also von E nach H) entspricht.

Shannons Modell eines klassischen Kryptosystems

Von Claude Shannon wurde folgendes Modell vorgeschlagen.

- Ein *Kryptosystem* ist ein Tripel (M, K, C) mit
  - Zufallsvariable (Nachricht) M
  - Zufallsvariable (Schlüssel) K
  - Zufallsvariable (Chiffre) C
- Die Ver- und Entschlüsselungsfunktionen E und D sind bekannt.
- Es gilt: $E(M, K) = C$ und $D(C, K) = M$.
- Insofern sind die drei Zufallsvariablen nicht unabhängig.
  - Wenn M und K gegeben ist, so steht C eindeutig fest.
  - Wenn C und K gegeben sind, so steht M eindeutig fest.

Shannons Modell eines klassischen Kryptosystems

Die weiteren Modellannahmen sind:

- Die Verteilung von M entspricht den Buchstabenhäufigkeiten der (z.B.) deutschen Sprache.
- K wird gleichverteilt gewählt aus einer Menge $\mathcal{K}$ von zur Verfügung stehenden Schlüsseln.
- Die Zufallsvariablen M und K werden als unabhängig angenommen.

Ein solches Kryptosystem (M, K, C) mit den Funktionen E, D gilt nach Shannon als absolut sicher, wenn die Zufallsvariablen M und C stochastisch unabhängig sind.

Intuitiv heißt das, dass aus der Kenntnis der Chiffre C keinerlei Rückschlüsse auf die Nachricht M gezogen werden können.

One-Time-Pad

Ein *One-Time-Pad* (auch: Vernam-Chiffre) ist ein Kryptosytem, das folgende Eigenschaften erfüllt:

- Der Schlüssel wird nur einmal verwendet, um die betreffende anstehende Nachricht zu verschlüsseln.
- Die Grundmengen der möglichen Nachrichten, der möglichen Schlüssel und der möglichen Chiffren haben dieselbe Mächtigkeit.
- Es soll gelten, dass für jeden Schlüssel k die Abbildung $m \mapsto E(m, k) = c$ eine bijektive Abbildung ist.
- Umgekehrt soll gelten, dass gegeben m und c der dazu gehörige Schlüssel k mit $c = E(m, k)$ eindeutig ist.
- Diesen bezeichnen wir mit $k(m, c)$.

Jedes one-time-pad-Kryptosystem ist absolut sicher

Es gilt: $P(M = m, C = c) = P(M = m, K = k(m, c))$

$$= P(M = m) \cdot P(K = k(m, c)) = P(M = m) \cdot \frac{1}{|\mathcal{K}|} \quad (1)$$

Außerdem gilt:

$$\begin{aligned} P(C = c) &= \sum_m P(M = m, K = k(m, c)) \\ &= \sum_m P(M = m) \cdot P(K = k(m, c)) \\ &= \sum_m P(M = m) \cdot \frac{1}{|\mathcal{K}|} \\ &= \frac{1}{|\mathcal{K}|} \cdot \sum_m P(M = m) = \frac{1}{|\mathcal{K}|} \quad (2) \end{aligned}$$

Also sind M und C unabhängig, denn

$$P(M = m, C = c) \overset{(1)}{=} P(M = m) \cdot \frac{1}{|\mathcal{K}|} \overset{(2)}{=} P(M = m) \cdot P(C = c)$$

One-Time-Pad

Rechnet man mit Bits 0 und 1, so ergibt sich ein one-time-pad, wenn

- man als Schlüssel so viele unabhängige und gleichverteilte Zufallsbits $k = (k_1, k_2, \dots)$ verwendet wie die Nachricht $m = (m_1, m_2, \dots)$ lang ist
- und mittels XOR die ebensolange Chiffre-Bitfolge erzeugt:

$$c = (c_1, c_2, \cdots) = (m_1 \oplus k_1, m_2 \oplus k_2, \dots) = m \oplus k$$

Der Schlüssel hängt dann eindeutig von Klartext und Chiffre ab: $k = m \oplus c$ (bitweise).

One-Time-Pad

Rechnet man modulo 26, so wäre ein one-time-pad

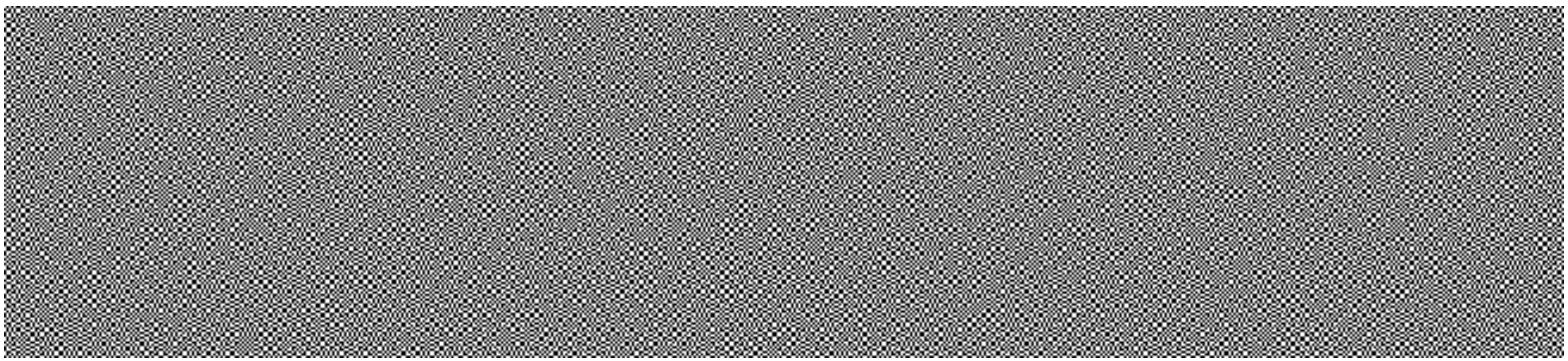
- eine Vigenère-Chiffre, bei der das Schlüsselwort ebenso lang ist wie der Klartext,
- wobei der Vigenère-Schlüssel nur einmal, für eben diesen Klartext, verwendet wird und
- die Buchstaben des Schlüssels unabhängig und gleichverteilt gewählt werden

Es ist klar, dass das Konzept des one-time-pads eine Idealvorstellung ist und nur schwer logistisch realisierbar ist.

Visuelle Kryptographie

Der Schlüssel ist ein Bild auf einer Folie, deren Pixel zufällig entweder von der Art links oder von der Art rechts ausgewählt werden:

Es entsteht ein Bild, das von weitem in einem einheitlichen Grauton erscheint:



Visuelle Kryptographie

Der Klartext (in Form eines Bilds) wird beim Verschlüsseln E so mit dem Schlüssel kombiniert,

- dass an bestimmten Stellen im Bild gerade die andere Pixelart gewählt wird (was später beim Übereinanderlegen eine Schwarzfärbung bewirkt),
- an anderen Stellen bleibt das Pixel unverändert so wie auf der Schlüsselfolie (es bleibt dann beim Übereinanderlegen bei einer Graufärbung).

Solcherart entsteht ein Chiffrebild, das für sich betrachtet ebenfalls zufällig grau gefärbt erscheint. Die Dechiffrierung D erfolgt einfach durch präzises Übereinanderlegen der beiden Folien.

Visuelle Kryptographie

Man kann bei der Visuellen Kryptographie leicht erkennen, dass die Schlüsselfolie nur für eine Nachricht verwendet werden sollte.

Legt man 2 mit derselben Folie verschlüsselte Folien übereinander (ohne die Schlüsselfolie zu besitzen), so entsteht eine Überlagerung beider Geheimnisse, so dass man unter Umständen beide Bilder rekonstruieren kann.

Klassische Kryptographie

Die historischen Kryptosysteme sind im Prinzip alle geknackt.
Es gibt aber heutzutage symmetrische Verfahren, die als sicher gelten, z.B.:

- Blum-Blum-Shub-Generator
- AES (Advanced Encryption Standard)

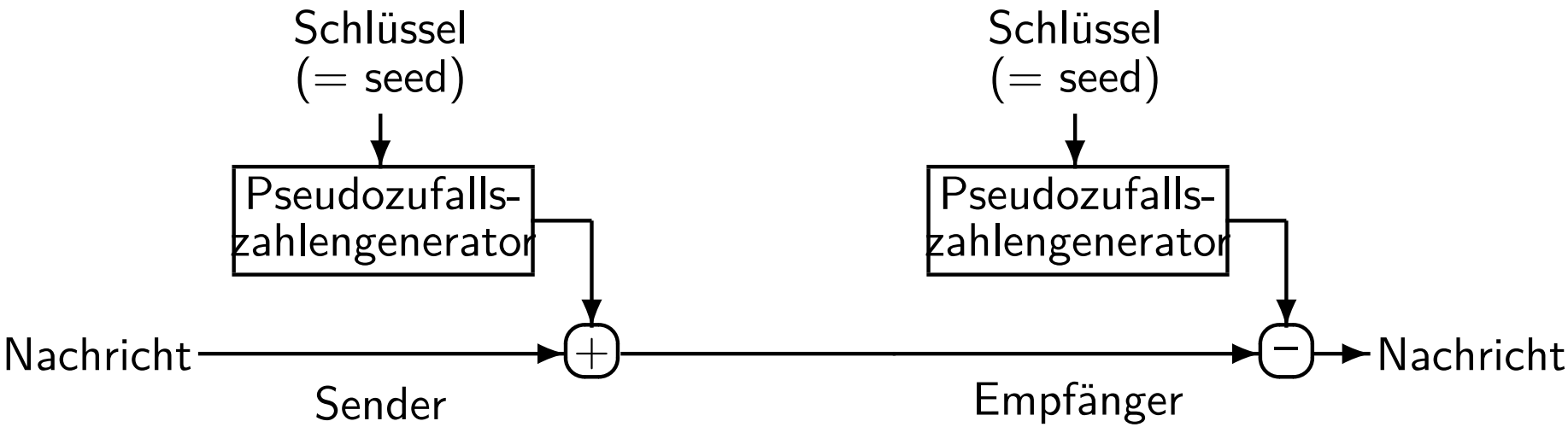
Diese basieren u.a. auf

- Pseudozufallsgeneratoren
- rückgekoppelten Schieberegistern

Pseudozufallsgeneratoren

- Man möchte Schlüssellängen haben, die so lang sind, dass es praktisch nicht möglich ist, alle Schlüssel durchzuprobieren.
- Viel länger als diese notwendige Mindestlänge sollten Schlüssel nicht sein, und dann aber für beliebig lange Nachrichten verwendbar sein.
- Idee: Pseudozufallszahlengeneratoren verwenden.
- Diese erzeugen aus einer kurzen Zahlen- oder Bit-Folge (dem seed, welcher hier die Rolle des Schlüssels übernimmt) eine sehr viel längere Zahlen- oder Bit-Folge, die möglichst alle statistischen Tests auf Zufälligkeit besteht (Gleichverteilung, Unabhängigkeit u.a.).
- Sender und Empfänger haben denselben seed zur Verfügung und können so dieselbe Zufallszahlenfolge (für die Ver- oder Entschlüsselung) erzeugen.

Pseudozufallsgeneratoren in der Kryptographie



Linearer Kongruenzgenerator

Ein gängiger Pseudozufallsgenerator, der Zahlen aus $\mathbb{Z}_n$ liefert, ist der lineare Kongruenzgenerator

- Man beginnt mit einer beliebigen Zahl $z_0 < n$ (dem seed) und rechnet dann $z_{i+1} = (a \cdot z_i + b) \bmod n$.
- Aus einem seed von k Bits entstehen so bis zu $n \approx 2^k$ viele Nachfolgezahlen, die zufällig wirken, bis sich die Folge wiederholt.
- Die maximale Periodenlänge von n wird genau dann erreicht, wenn für a, b, n Folgendes gilt:
  - b und n sind teilerfremd.
  - $a \equiv 1 \pmod{p}$ für alle Primteiler p von n.
  - Falls n durch 4 teilbar ist, muss $a \equiv 1 \pmod{4}$ gelten.
  - Beweis siehe z.B. in Knuth: The Art of Computer Programming, Vol. 2, Theorem A.
- Ein linearer Kongruenzgenerator ist aus kryptographischer Sicht nicht sicher.

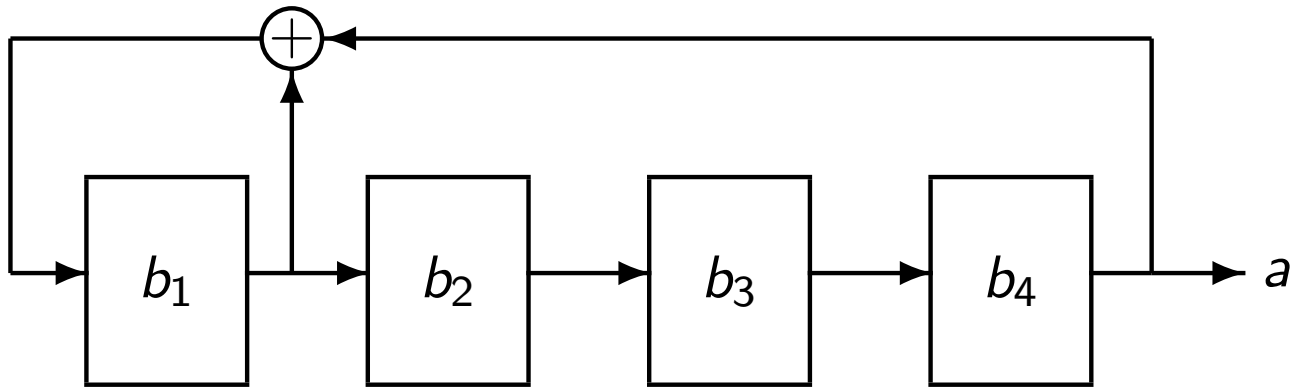
Blum-Blum-Shub-Generator

- Besser ist es, wenn man Polynome höheren Grades verwendet, etwa
$$z_{i+1} = (a \cdot z_i^2 + b \cdot z_i + c) \bmod n$$
- Der Blum-Blum-Shub-Generator verwendet die Rekursion $z_{i+1} = z_i^2 \bmod n$ und als Modul eine Zahl $n = p \cdot q$, wobei p und q Blum-Primzahlen sind.
- Eine Primzahl p heißt Blum-Primzahl, wenn $p + 1$ durch 4 teilbar ist, d.h. $p \equiv 3 \pmod 4$.
- Für den Seed z_0 muss gelten: $\mathrm{ggT}(z_0, n) = 1$.

Beispiel: Sei $n = 23 \cdot 29$ und $z_0 = 2$. Dann ergeben sich folgende Pseudozufallszahlen: 2, 4, **16**, 256, 170, 219, 604, 634, 422, 662, 25, 625, 430, 141, 538, 633, 489, 335, 169, 547, 393, 372, 315, 509, 285, 518, 190, 82, 54, 248, 140, 257, **16**.
Von nun ab wiederholt sich die Folge. Die Periodenlänge ist 30. Die Vorperiode hat die Länge 3.

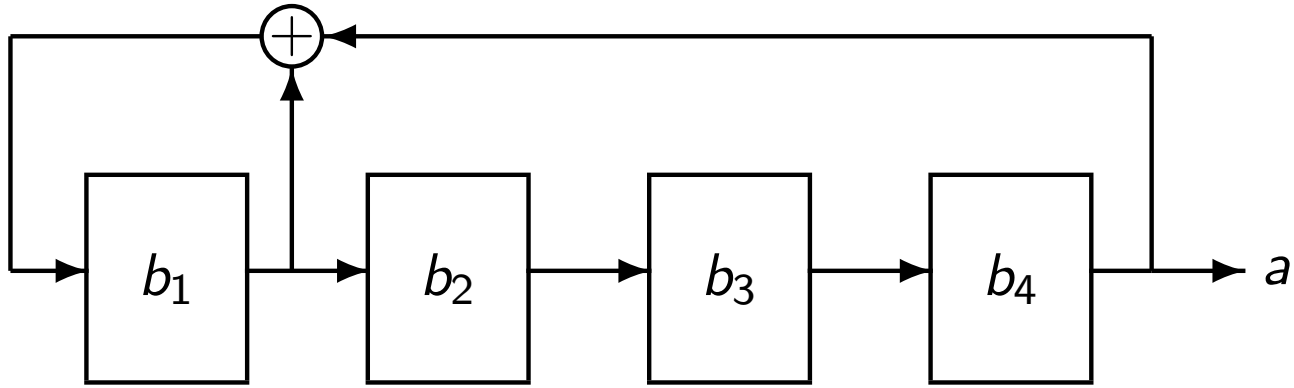
Rückgekoppelte Schieberegister

Rückgekoppelte Schieberegister sind technisch einfach realisierbare Pseudozufallsgeneratoren, die sehr lange zufällig wirkende Bit-Folgen als Ausgabe produzieren, ehe sich die Folge wiederholt.



- Das Beispiel zeigt ein Schieberegister bestehend aus 4 Flip-Flips, dies sind Speicher für jeweils 1 Bit (0 oder 1).
- Mit jedem Taktsignal wird der 4-Bit-Inhalt um eine Position nach rechts verschoben.
- Das letzte Flip-Flop liefert das Ausgabesignal.

Rückgekoppelte Schieberegister



Eine Rückkopplungsschaltung, bestehend aus einem oder mehreren XOR-Gattern verknüpft die Ausgaben einiger Flip-Flops des Schieberegisters (insbesondere muss das letzte Flip-Flop dabei beteiligt sein) und führt das Signal auf das erste Flip-Flop zurück.

Startet man bei diesem Beispiel mit der Anfangskonfiguration 1111 (das wäre der seed), so durchläuft das Schieberegister folgende Konfigurationen: 0111, 1011, 0101, 1010, 1101, 0110, 0011, 1001, 0100, 0010, 0001, 1000, 1100, 1110. Danach wiederholt sich die Folge.

AES (Advanced Encryption Standard)

- AES ist eine Block-Chiffre mit Blocklänge 128 Bit und Schlüssellängen 128, 192 oder 256 Bit.
- AES ist ein vom National Institue of Standards and Technology (NIST) ausgewählter Standard, der heutigen kryptographischen Ansprüchen genügt.
- Der Schlüssel wird in mehrere Rundenschlüssel expandiert (ähnlich den Methoden bei Pseudozufallsgeneratoren).
- In mehreren, aufeinanderfolgenden Runden (10 bei Schlüssellänge 128), in die jeweils einer der Rundenschlüssel einfließt, wird der zu verschlüsselnde Block komplexen Substitutionen und Transpositionen unterworfen.

Euklid-Algorithmus

Rekursive Formulierung:

```
PROC ggT( a, b )
        IF (b == 0) THEN RETURN a
        ELSE RETURN ggT( b, a mod b )
```

- Wir gehen o.B.d.A. davon aus, dass $a \geq b$. Sollte a kleiner sein als b, so werden a und b beim ersten rekursiven Aufruf sowieso vertauscht.
- Die Umformung von (a, b) zu $(b, a \bmod b)$ ist korrekt, denn es gilt, dass die Menge der gemeinsamen Teiler von a und b identisch ist mit der Menge der gemeinsamen Teiler von b und $(a \bmod b)$. Daher stimmen auch deren größte Elemente überein.

Euklid-Algorithmus

Von Hand führt man Zeile für Zeile eine ganzzahlige Division von a durch b mit Rest durch und in der nachfolgenden Zeile übernimmt b die Rolle von a und der Rest die Rolle von b, usw. bis sich Rest $= 0$ ergibt.

Beispiel: Eingabe sei $a = 102$ und $b = 42$.

$$\begin{array}{rcrcr} 102 & = & 2 \cdot 42 & \text{Rest} & 18 \\ 42 & = & 2 \cdot 18 & \text{Rest} & 6 \\ 18 & = & 3 \cdot 6 & \text{Rest} & 0 \end{array}$$

Der ggT von 102 und 42 ist also 6.

Euklid-Algorithmus: Laufzeit

Seien x und y natürliche Zahlen mit Bitlänge m und n mit $m \geq n$. Sei k die Bitlänge des Quotienten q der Division von x durch y, d.h. $x = q \cdot y + r$ (ganzzahlige Division mit Rest $r \in \mathbb{N}$, $0 \leq r < y$).

- Man kann $(x + y)$ sowie $(x - y)$ mit Rechenaufwand $O(m)$ berechnen.
- Die übliche Schulmethode benötigt für die Multiplikation $x \cdot y$ die Zeit $O(m \cdot n)$.
- Die Berechnung von x div y und auch von x mod y benötigt mit der Schulmethode $O(k \cdot n)$ Aufwand. Da der Quotient q die Bitlänge $k = O(m - n)$ hat, benötigt die Division die Rechenzeit $O((m - n) \cdot n)$.

Euklid-Algorithmus: Laufzeit

- Seien die im Verlauf von ggT entstehenden Zahlen $a_1 \geq a_2 \geq a_3 \geq \cdots \geq a_t > 0 = a_{t+1}$.
- Hierbei sind a_1, a_2 die Eingabezahlen und a_t ist der berechnete größte gemeinsame Teiler.
- Die betreffenden Bitlängen seien $k_1 \geq k_2 \geq k_3 \geq \cdots \geq k_t \geq 1$.
- Die jeweils durchzuführenden Divisionen mit Rest benötigen einen Rechenaufwand in der Größenordnung

$$
\begin{aligned}
& (k_1 - k_2)k_2 + (k_2 - k_3)k_3 + (k_3 - k_4)k_4 + \cdots \\
= \; & k_1 k_2 - k_2^2 + k_2 k_3 - k_3^2 + k_3 k_4 - k_4^2 + - \cdots \\
\leq \; & k_1 k_2 - k_2 k_3 + k_2 k_3 - k_3 k_4 + k_3 k_4 - k_4 k_5 + - \cdots
\end{aligned}
$$

- In dieser Teleskopsumme bleiben nur der erste und der letzte (konstante) Term übrig, daher ergibt sich ein Rechenaufwand von $O(k_1 k_2)$.

Erweiterter Euklid-Algorithmus

Außer dem größten gemeinsamen Teiler $d = \mathrm{ggT}(a,b)$ werden noch zwei ganze Zahlen x, y zurückgeliefert, so dass $ax + by = d$.

```
PROC extggT( a, b )
        IF (b == 0) THEN RETURN (a,1,0);
        (d,x,y) := extggT( b, a mod b );
        RETURN ( d, y, x − (a div b) ∗ y )
```

Verfolgt man die rekursiven Aufrufe von extggT bei Eingabe $(102, 42)$, so erhält man folgende Zwischenergebnisse:

a	b	d	x	y
102	42	6	-2	5
42	18	6	1	-2
18	6	6	0	1
6	0	6	1	0

Im linken Teil der Tabelle erfolgt der rekursive Abstieg, der auf das Ergebnis $\mathrm{ggT} = 6$ führt, mit den Werten auf der rechten Seite kehrt die Rekursion schließlich zurück ins aufrufende Programm.

Erweiterter Euklid-Algorithmus

- Bei der Rechnung von Hand führt man zunächst eine ggT-Berechnung durch wie beim Beispiel oben.
- Beginnend mit der vorletzten Zeile kann man Zeile für Zeile von unten nach oben einsetzen, bis man die Darstellung des ggT als Linearkombination von a und b erreicht.
- Die Laufzeit von extggT ist wie bei ggT ebenfalls $O(k_1 k_2)$.

Beispiel: Eingabe sei $a = 102$ und $b = 42$.

$$\begin{aligned} 102 &= 2 \cdot 42 + 18 \\ 42 &= 2 \cdot 18 + 6 \\ 18 &= 3 \cdot 6 + 0 \end{aligned}$$

Der ggT ist also 6. Beginnend mit der vorletzten Zeile, umgeformt als $6 = 42 - 2 \cdot 18$, ersetzt man die 18 mit Hilfe der umgeformten ersten Zeile $18 = 102 - 2 \cdot 42$ und erhält

$$6 = 42 - 2 \cdot 18 = 42 - 2 \cdot (102 - 2 \cdot 42) = (-2) \cdot 102 + 5 \cdot 42$$

Also ist $x = -2$ und $y = 5$.

Modulare Kongruenzrelation und mod-Operation

- Die Notation $x \equiv y \pmod{n}$ bedeutet, dass $x - y$ durch n teilbar ist.
- Dies ist eine Äquivalenzrelation auf $\mathbb{Z}$ (den ganzen Zahlen).
- Das heißt, $\mathbb{Z}$ wird in n Äquivalenzklassen zerlegt:

$$\mathbb{Z} = [0] \cup [1] \cup \cdots \cup [n-1] \quad \text{wobei} \quad [x] = \{\, y \in \mathbb{Z} \mid x \equiv y \pmod{n} \,\}$$

- Die Zahlen $0, 1, \ldots, n-1$ bilden ein vollständiges Repräsentantensytem, d.h. für alle $y \in \mathbb{Z}$ existiert genau ein $x \in \{0, 1, \ldots, n-1\}$ mit $x \equiv y \pmod{n}$.
- Wenn wir ab dem nächsten Abschnitt schreiben

$$\mathbb{Z}_n = \{0, 1, \ldots, n-1\} \quad \text{statt} \quad \mathbb{Z}_n = \{[0], [1], \ldots, [n-1]\}$$

so muss man diese vereinfachte Schreibweise immer so verstehen, dass $x \in \mathbb{Z}_n$ für eine ganze Äquivalenzklasse steht.

Modulare Kongruenzrelation und mod-Operation

- Programmiertechnisch ist es allerdings so, dass x mod n immer einen dieser kanonischen Repräsentanten aus der Menge $\{0, 1, \ldots, n-1\}$ liefert.
- Das heißt x mod n ergibt den Rest bei der Division von x durch n; dies ist eine Zahl zwischen 0 und $n-1$.
- Ferner bedeutet div das ganzzahlige Divisionergebnis (ohne Rest), also x div $y = \lfloor x/y \rfloor$.
- Es gilt: $x \equiv y \pmod{n}$ gdw. $(x \bmod n) = (y \bmod n)$.
- Außerdem gilt: $a \bmod b = a - (a \text{ div } b) \cdot b$

Gruppen und Körper

Die Menge $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, zusammen mit der modularen Addition ist eine kommutative Gruppe:

- es gilt das Assoziativgesetz
- es gilt das Kommutativgesetz
- es gibt ein neutrales Element, das ist die 0
- jedes Element $a \in \mathbb{Z}_n$ hat ein Inverses, nämlich $-a$ bzw. $n - a$, denn $a + (n - a) \equiv a + (-a) \equiv 0 \pmod{n}$

Diese Gruppe wird nur selten in der Kryptographie benötigt.

Additionstabelle für $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$:

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Gruppen und Körper

- Die Menge $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n-1,\ ggT(n,a) = 1\}$ mit der modularen Multiplikation als Rechenoperation ist ebenfalls eine kommutative Gruppe.
- Man spricht von einem reduzierten Restsystem bzw. der primen Restklassengruppe.
- Das neutrale Element ist die 1.
- Jedes Element $a \in \mathbb{Z}_n^*$ besitzt ein multiplikatives Inverses $a' \in \mathbb{Z}_n^*$ mit $aa' \equiv 1 \pmod{n}$.

Gruppen und Körper

- Da $\mathrm{ggT}(n, a) = 1$, liefert der erweiterte Euklid-Algorithmus $\mathrm{extggT}(n, a) = (1, x, y)$ mit $1 = n \cdot x + a \cdot y$. Rechnen modulo n ergibt: $1 \equiv n \cdot x + a \cdot y \equiv a \cdot y \pmod{n}$. Das heißt y mod n ist das multiplikativ inverse Element zu a.
- Es gilt $ggT(n, y) = 1$, d.h. $y \in \mathbb{Z}_n^*$.
- Für $n = 102$ und $a = 65$ liefert $\mathrm{extggT}(n, a) = (1, -7, 11)$, d.h. $1 = 102 \cdot -7 + 65 \cdot 11$. Das Inverse zu 65 ist also 11.

Gruppen und Körper

Multiplikationstabelle zu $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

$*$	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Gruppen und Körper

Beobachtungen zur Multiplikationstabelle von $\mathbb{Z}_{15}^*$:

- Die Tabelle ist symmetrisch zur Hauptdiagonalen, d.h. die Multiplikation ist kommutativ.
- In jeder Zeile/Spalte gibt es genau eine 1; also hat jedes Element genau ein Inverses.
- Jede Zeile/Spalte enthält alle Elemente; also ist die Abbildung $x \mapsto ax$ bijektiv auf $\mathbb{Z}_{15}^*$.
- Die obigen Eigenschaften gelten allgemein für $\mathbb{Z}_n^*$.
- Entlang der Hauptdiagonalen kommen nur die Zahlen 1 und 4 vor; also sind dies die einzigen Quadratzahlen (quadratischen Reste) modulo 15.
- Beispielsweise hat die 1 die vier Quadratwurzeln 1,4,11,14.

Gruppen und Körper

Wenn n Primzahl ist, dann ist $\mathbb{Z}_n$ zusammen mit der modularen Addition und der modularen Multiplikation ein Körper, denn

- $(\mathbb{Z}_n, +)$ ist eine kommutative Gruppe
- $(\mathbb{Z}_n \setminus \{0\}, *) = (\mathbb{Z}_n^*, *)$ ist eine kommutative Gruppe
- und es gelten die Distributivgesetze.

Chinesischer Restsatz

Gegeben sind paarweise teilerfremde Zahlen $n_1, \ldots, n_k$ sowie $a_1 \in \mathbb{Z}_{n_1}, \ldots, a_k \in \mathbb{Z}_{n_k}$.
Gesucht ist eine Lösung x für das Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

Beh.: Es existiert eine Lösung x, die eindeutig ist modulo $\prod_{i=1}^{k} n_i$.

Bem.: Die kryptographischen Anwendungen beschränken sich meist auf $k = 2$ und zwei Primzahlen $n_1 = p$, $n_2 = q$.

Chinesischer Restsatz

Die Lösung x lässt sich durch folgenden Algorithmus berechnen.

Berechne $n = n_1 \cdot n_2 \cdots n_k$
Berechne $m_1 = n/n_1$, $m_2 = n/n_2, \ldots$, $m_k = n/n_k$.

- Es gilt $m_i \equiv 0(\mod n_j)$ für $j \neq i$, weil $m_i = \prod_{j \neq i} n_j$
- Da $ggT(m_i, n_i) = 1$, existiert ein y_i invers zu m_i modulo n_i.

Bestimme die entsprechenden multiplikativen Inversen: y_1 invers zu m_1 modulo n_1, usw. bis y_k invers zu m_k modulo n_k. Gib $x = \left(\sum_{i=1}^{k} a_i \cdot y_i \cdot m_i\right) \mod n$ aus.

Die Ausgabe x ist Lösung des Kongruenzgleichungssystems, denn

$$\begin{aligned} a_i \cdot y_i \cdot m_i &\equiv a_i \cdot 1 \equiv a_i \pmod{n_i} \text{ für } 1 \leq i \leq k \\ a_j \cdot y_j \cdot m_j &\equiv 0 \pmod{n_i} \text{ für } j \neq i \\ x &\equiv a_i \cdot y_i \cdot m_i + \sum_{j \neq i} a_j \cdot y_j \cdot m_j \equiv a_i \pmod{n_i} \end{aligned}$$

Chinesischer Restsatz: Zahlenbeispiel

Gesucht ist eine Lösung x, die folgende Kongruenzgleichungen erfüllt:

$$\begin{aligned} x &\equiv 4 \pmod 9 \\ x &\equiv 3 \pmod 8 \\ x &\equiv 0 \pmod 7 \end{aligned}$$

Da die Zahlen 9,8,7 paarweise teilerfremd sind, kann der Chinesische Restsatz angewandt werden. Wir berechnen:
$n = 9 \cdot 8 \cdot 7 = 504$, $m_1 = 504/9 = 56$, $m_2 = 504/8 = 63$, $m_3 = 504/7 = 72$.
Nun müssen die multiplikativen Inversen von 56, 63, 72 bestimmt werden, und zwar modulo 9, modulo 8 und modulo 7. Zunächst stellen wir fest, dass

$$\begin{aligned} 56 &\equiv 2 \pmod 9 \\ 63 &\equiv 7 \pmod 8 \\ 72 &\equiv 2 \pmod 7 \end{aligned}$$

Chinesischer Restsatz: Zahlenbeispiel

Im Allgemeinen müssten wir zur Inversenbestimmung den erweiterten Euklid-Algorithmus anwenden.
Wir bestimmen die Inversen modulo 9 bzw. 8 bzw. 7 von Hand:

$$\begin{aligned}
&\text{Inverse mod } 9: && (1,1)\,(2,5)\,(4,7)\,(5,2)\,(7,4)\,(8,8)\\
&\text{Inverse mod } 8: && (1,1)\,(3,3)\,(5,5)\,(7,7)\\
&\text{Inverse mod } 7: && (1,1)\,(2,4)\,(3,5)\,(4,2)\,(5,3)\,(6,6)
\end{aligned}$$

Hier können wir nun ablesen, dass $y_1 = 5$, $y_2 = 7$, $y_3 = 4$. In die Formel für x eingesetzt ergibt dies

$$x = (4 \cdot 5 \cdot 56 + 3 \cdot 7 \cdot 63 + 0 \cdot 4 \cdot 72) \bmod 504 = 427$$

Innerhalb der Zahlen $\mathbb{Z}_{504} = \{0, 1, \ldots, 503\}$ ist diese Lösung eindeutig. Außerhalb dieses Zahlenbereichs wiederholen sich die Lösungen im Abstand von jeweils 504.

Chinesischer Restsatz

Gegeben seien paarweise teilerfremde Zahlen $n_1, \ldots, n_k$.
Wir zeigen, dass die Abbildung

$$\begin{array}{rcl} \chi : \mathbb{Z}_n & \to & \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k} \\ x \bmod n & \mapsto & (x \bmod n_1, \ldots, x \bmod n_k) \end{array}$$

bijektiv ist, wobei $n = n_1 \cdot n_2 \cdots n_k$.

Die Surjektivität ist klar, denn für $(a_1, \ldots, a_k) \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$ existiert mit dem Chinesischen Restsatz ein $x \in \mathbb{Z}_n$ mit $\chi(x) = (a_1, \ldots, a_k)$.

Die Bijektivität folgt nun daraus, dass die beiden Mengen gleich mächtig sind: $|\mathbb{Z}_n| = n = n_1 \cdot n_2 \cdots n_k = |\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}|$.

Chinesischer Restsatz

Alternativer Beweis: Um die Injektivität nachzuweisen, müssen wir zeigen, dass für $x, x' \in \mathbb{Z}_n$ aus $\chi(x) = \chi(x')$ die Kongruenz $x \equiv x' \pmod{n}$ folgt.

- Seien also x und x' Lösungen des obigen Kongruenzgleichungssystems, d.h. $\chi(x) = (a_1, \ldots, a_k) = \chi(x')$.
- Dann gilt $x \equiv x' \pmod{n_i}$ und daher ist $x - x'$ durch n_i teilbar für alle i mit $1 \leq i \leq k$.
- Da $n_1, \ldots, n_k$ paarweise teilerfremde Zahlen sind, ist $x - x'$ durch n teilbar und damit $x \equiv x' \pmod{n}$.

Chinesischer Restsatz: Beispiel

- Die Zahlen $n_1 = 5$ und $n_2 = 6$ sind teilerfremd.
- Das Ausfüllen einer 5×6 großen Tabelle veranschaulicht die Korrespondenz zwischen $\mathbb{Z}_5 \times \mathbb{Z}_6$ und $\mathbb{Z}_{30}$.
- Man fängt im linken oberen Eck mit 0 an, dann $1, 2, 3, \ldots$ diagonal nach rechts unten mit wrap-around in die Tabelle eintragen.
- Die Aussage des CR ist dabei, dass alle Tabellenelemente ohne Lücke ausgefüllt werden.

	0	1	2	3	4	5
0	0	25	20	15	10	5
1	6	1	26	21	16	11
2	12	7	2	27	22	17
3	18	13	8	3	28	23
4	24	19	14	9	4	29

Chinesischer Restsatz

Sei $n = n_1 n_2 \cdots n_k$, wobei $n_1, \ldots, n_k$ paarweise teilerfremd sind.

Es gilt $\mathrm{ggT}(n, x) = 1$ gdw. $\mathrm{ggT}(n_i, x) = 1$ für alle i mit $1 \leq i \leq k$.

Daher ist χ beschränkt auf $\mathbb{Z}_n^*$ eine bijektive Abbildung zwischen $\mathbb{Z}_n^*$ und $\mathbb{Z}_{n_1}^* \times \cdots \times \mathbb{Z}_{n_k}^*$.

Für die Umkehrung von χ benutzen wir die Notation $(a_1, a_2, \ldots, a_k) \circ\!\!-\!\!\bullet\, x$.

Chinesischer Restsatz: Beispiel

Indem man sich auf die reduzierten Restklassen einschränkt, erhält man die Korrespondenz zwischen $\mathbb{Z}_5^* \times \mathbb{Z}_6^* = \{1,2,3,4\} \times \{1,5\}$ und $\mathbb{Z}_{30}^* = \{1,7,11,13,17,19,23,29\}$.

	1	5
1	1	11
2	7	17
3	13	23
4	19	29

Die Euler-Funktion φ

- Definition: $\varphi(n) = |\{a \mid 1 \leq a \leq n,\ ggT(n,a) = 1\}|$
- Die Definition beinhaltet $\varphi(1) = 1$.
- Es gilt $\varphi(n) = |\mathbb{Z}_n^*|$.
- Für Primzahlen n gilt $\mathbb{Z}_n^* = \{1, 2, \ldots, n-1\}$.
- Für Primzahlen n gilt also $\varphi(n) = n - 1$.

Die Euler-Funktion φ

Sei $n = \prod_{i=1}^{k} (p_i)^{e_i}$ die Primzahlzerlegung von n. Dann gilt

$$\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{e_i}) = \prod_{i=1}^{k} (p_i - 1) \cdot (p_i)^{e_i - 1} = n \cdot \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

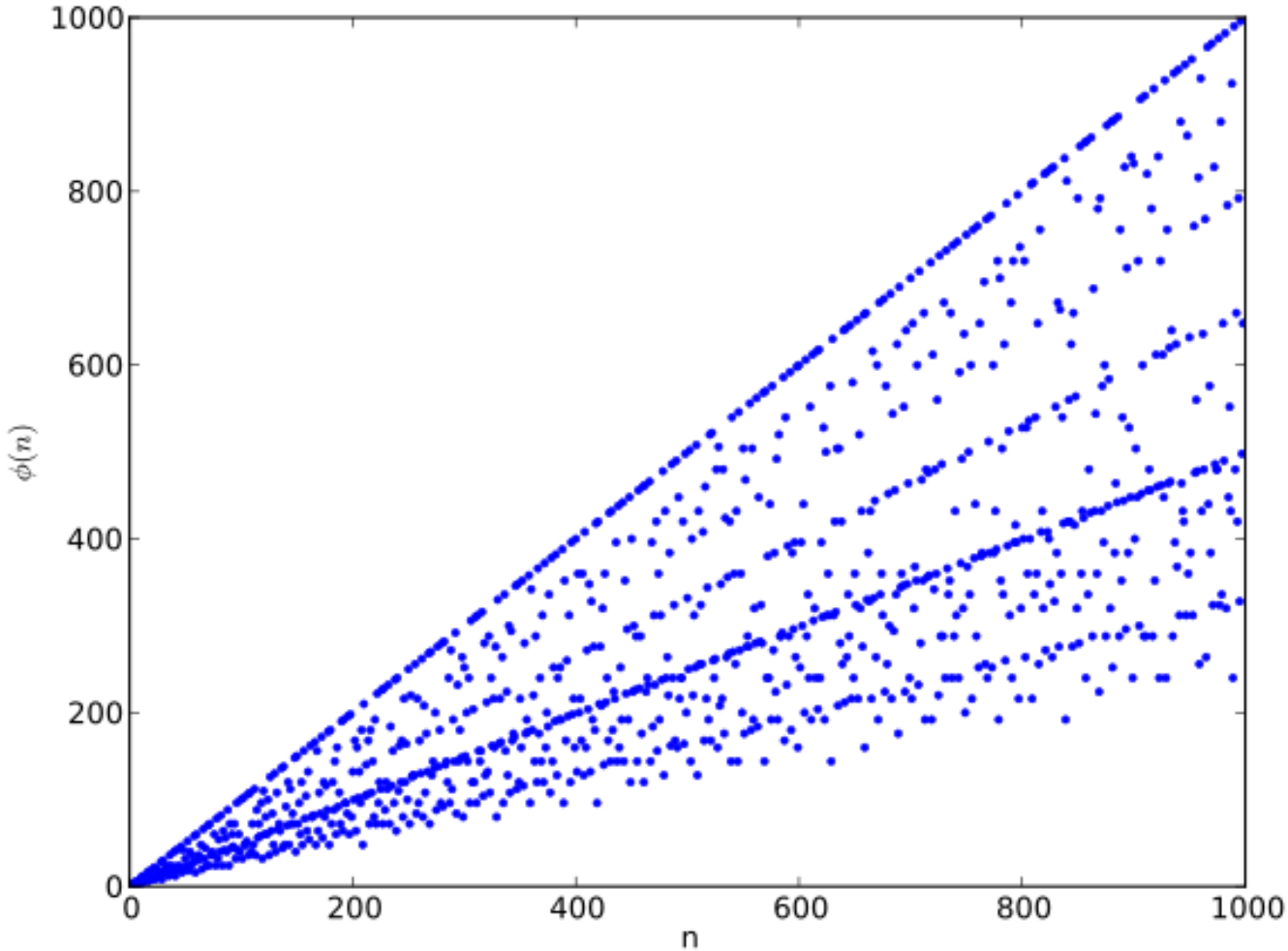
Beweis:

- Es gilt $\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{e_i})$ wegen der Isomorphie $\mathbb{Z}_n^* \to \mathbb{Z}_{p_1^{e_1}}^* \times \cdots \times \mathbb{Z}_{p_k^{e_k}}^*$.
- Wir berechnen $\varphi(p^e)$ für eine Primzahl p und den Exponenten $e \geq 1$.
- Es gilt $\mathbb{Z}_{p^e}^* = \{a \mid 1 \leq a \leq p^e - 1, ggT(a, p^e) = 1\}$, d.h. genau die Vielfachen von p sind nicht Element von $\mathbb{Z}_{p^e}^*$.
- Von denen gibt es p^{e-1} viele, also folgt

$$\varphi(p^e) = p^e - p^{e-1} = (p-1)p^{e-1} = p^e \left(1 - \frac{1}{p}\right)$$

Bem.: Für das Produkt $n = p \cdot q$ zweier Primzahlen p und q gilt somit $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$.

Euler-Funktion $\varphi(n)$ für $n = 1, \ldots, 1000$



Zahlenbeispiel: Sei $n = 2^3 \cdot 5^2 \cdot 11 \cdot 13^4 = 62834200$. Dann ist $\varphi(n) = 1 \cdot 2^2 \cdot 4 \cdot 5 \cdot 10 \cdot 12 \cdot 13^3 = 21091200$.

$\mathbb{Z}_n^*$ ist eine kommutative Gruppe

- Die Menge $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n-1,\ ggT(n,a) = 1\}$ mit der modularen Multiplikation ist eine kommutative Gruppe.
- Für $a \in \mathbb{Z}_n^*$ und $k \in \mathbb{N}$ sei a^k definiert durch

$$a^k = \underbrace{a \cdot a \cdots a}_{k\text{-mal}} \mod n$$

- Für $a \in \mathbb{Z}_n^*$ bezeichnet $a^{-1} \in \mathbb{Z}_n^*$ das inverse Element von a.
- Für $a \in \mathbb{Z}_n^*$ und $k \in \mathbb{N}$ definieren wir $a^{-k} = (a^{-1})^k$.
- Bem.: a^{-k} ist das inverse Element von a^k.
- Es gilt $a^\ell \cdot a^m \equiv a^{\ell+m} \pmod{n}$ für alle $\ell, m \in \mathbb{Z}$.
- Es gilt $(a^\ell)^m \equiv a^{\ell \cdot m} \pmod{n}$ für alle $\ell, m \in \mathbb{Z}$.
- Für $a, b \in \mathbb{Z}_n^*$ und $m \in \mathbb{Z}$ gilt $(a \cdot b)^m \equiv a^m \cdot b^m \pmod{n}$.

Multiplikationstabelle zu $\mathbb{Z}_{15}^*$

Multiplikationstabelle zu $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

$*$	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

Ordnung eines Elements

- Sei $a \in \mathbb{Z}_n^*$. In der Folge $1 = a^0, a = a^1, a^2, a^3, \ldots$ muss sich irgendwann eine Periode ergeben.
- Sei k der kleinste Exponent, so dass $a^k = a^i$ für ein $i < k$.
- Wenn man annimmt, dass $i > 0$, so kann man die Gleichung mit dem Inversen von a multiplizieren und erhält $a^{k-1} = a^{i-1}$.
- Also kann k nicht der kleinste solche Exponent gewesen sein.
- Es folgt, dass $i = 0$ sein muss, also $a^k = a^0 = 1$.
- Ab diesem Exponenten wiederholt sich die Folge periodisch: $a^1 = a^{k+1}, a^2 = a^{k+2}, \ldots$.
- Den kleinsten Exponenten $k > 0$ mit $a^k = 1$ nennt man die *Ordnung* (oder Periode) des Elements a.

Satz von Euler

Satz von Euler: Für jedes $a \in \mathbb{Z}_n^*$ gilt: $a^{\varphi(n)} \equiv 1 \pmod{n}$.
Beweis:

- Da $\mathbb{Z}_n^*$ eine Gruppe ist, folgt aus $a \cdot x \equiv a \cdot y \pmod{n}$, dass $x \equiv y \pmod{n}$.
- Multipliziert man also a mit jedem einzelnen Element von $\mathbb{Z}_n^*$, so entsteht wieder die gesamte Menge $\mathbb{Z}_n^*$.
- Das heißt $a \cdot \mathbb{Z}_n^* = \{a \cdot x \mid x \in \mathbb{Z}_n^*\} = \mathbb{Z}_n^*$.
- Also gilt (modulo n):

$$\prod_{x \in \mathbb{Z}_n^*} (a \cdot x) \equiv \prod_{x \in \mathbb{Z}_n^*} x \quad \text{bzw.} \quad a^{\varphi(n)} \cdot \prod_{x \in \mathbb{Z}_n^*} x \equiv \prod_{x \in \mathbb{Z}_n^*} x$$

- Indem man beide Seiten der Gleichung mit dem Inversen von $\prod_{x \in \mathbb{Z}_n^*} x$ multipliziert, erhält man $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Korollar zum Satz von Euler

Theorem: Es sei k die Ordnung des Elements $a \in \mathbb{Z}_n^*$ und $e \in \mathbb{Z}$. Dann gilt $a^e \equiv 1 \pmod{n}$ gdw. k ein Teiler von e ist.

Bew.: Wenn $e = k\ell$ ist, folgt $a^e \equiv a^{k\ell} \equiv (a^k)^\ell \equiv 1^\ell \equiv 1 \pmod{n}$. Sei umgekehrt $a^e \equiv 1 \pmod{n}$.

- Sei $e = qk + r$ mit $0 \leq r < k$.
- Dann folgt $a^r \equiv a^{e-qk} \equiv a^e(a^k)^{-q} \equiv 1 \pmod{n}$.
- Weil k die kleinste Zahl > 0 ist mit $a^k \equiv 1 \pmod{n}$, und weil $0 \leq r < k$ gilt, muss $r = 0$ und damit $e = qk$ sein. Also ist k ein Teiler von e.

Korollar: Die Ordnung eines Elements $a \in \mathbb{Z}_n^*$ ist ein Teiler von $\varphi(n)$ (Kombination aus obigem Theorem und dem Satz von Euler).

Weiteres wichtiges Korollar

Korollar: Es sei k die Ordnung des Elements $a \in \mathbb{Z}_n^*$ und $\ell, m \in \mathbb{Z}$. Dann gilt

$$a^\ell \equiv a^m \pmod{n} \qquad \text{gdw.} \qquad \ell \equiv m \pmod{k}$$

Beweis: Setze $e = \ell - m$ und wende obiges Theorem an, d.h. es gilt $a^{\ell-m} \equiv 1 \pmod{n}$ gdw. k ein Teiler von $\ell - m$ ist.

Korollar zum Satz von Euler

Satz: Für $a \in \mathbb{Z}_n$ sind die folgenden Aussagen äquivalent:

(i) $a \in \mathbb{Z}_n^*$, d.h. a und n sind teilerfremd.

(ii) $a^{\varphi(n)} \equiv 1 \pmod{n}$.

(iii) a besitzt ein multiplikatives Inverses modulo n.

Beweis:

- Aus (i) folgt (ii): das ist der Satz von Euler.
- Aus (ii) folgt (iii): das Inverse von a ist $a^{\varphi(n)-1}$.
- Aus (iii) folgt (i): Wenn $a \in \mathbb{Z}_n$ ein multiplikatives Inverses $a' \in \mathbb{Z}_n$ besitzt, so gilt $aa' - 1 \equiv 0 \pmod{n}$, also $aa' - 1 = kn$ für eine Zahl k. Wenn nun d ein gemeinsamer Teiler von a und n ist, also $a = da_1$ und $n = dn_1$, so folgt

$$1 = a \cdot a' - k \cdot n = d \cdot a_1 \cdot a' - k \cdot d \cdot n_1 = d \cdot (a_1 a' - k n_1)$$

Da $d \in \mathbb{Z}$, folgt $|d| = 1$ und damit $\mathrm{ggT}(a, n) = 1$, also $a \in \mathbb{Z}_n^*$.

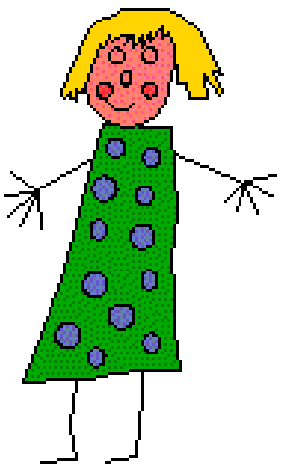
Satz von Fermat

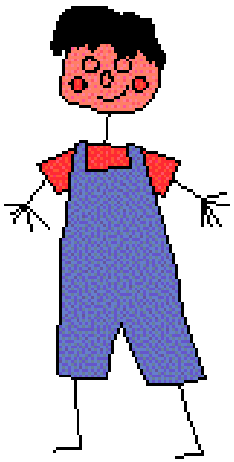
- Wenn n Primzahl ist, so ist $\varphi(n) = n - 1$.
- Man kann dann alle oben angegebenen Aussagen mit $n - 1$ statt $\varphi(n)$ schreiben.
- Insbesondere nennt man die Aussage, dass $a^{n-1} \equiv 1 \pmod{n}$ für alle $a \in \mathbb{Z}_n^*$ gilt, den kleinen Satz von Fermat.

RSA-Kryptosystem: Genereller Ablauf

- RSA steht für Rivest–Shamir–Adleman
- Alice will Bob eine verschlüsselte Nachricht senden, die nur Bob entschlüsseln kann.
- Bob erzeugt ein Schlüsselpaar, bestehend aus einem öffentlichen Schlüssel (n, e) und einem privaten (geheimen) Schlüssel d.
- Alice verwendet den öffentlichen Schlüssel (n, e), um ihre Nachricht zu verschlüsseln, und sendet die verschlüsselte Nachricht an Bob.
- Bob verwendet seinen privaten Schlüssel d, um die verschlüsselte Nachricht zu entschlüsseln.

RSA-Kryptosystem: Asymmetrische Kryptographie





Init

(n, e)

(n, e) d

m $c = m^e \bmod n$ $c^d \bmod n = m$

RSA: Schlüsselgenerierung

- Wähle zufällig zwei sehr große Primzahlen $p \neq q$ (jeweils ca. 500 Bit lang).
- Berechne $n = p \cdot q$ (n heißt RSA-Modul).
- Wähle eine zu $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$ teilerfremde Zahl e, für die gilt $1 < e < \varphi(n)$.
- Es muss also $ggT(e, \varphi(n)) = 1$ gelten.
- Das Paar (n, e) ist der öffentliche Schlüssel.
- Berechne eine natürliche Zahl d mit $1 < d < \varphi(n)$ und $d \cdot e \equiv 1 \pmod{\varphi(n)}$.
- d, das multiplikativ inverse Element zu e modulo $\varphi(n)$, ist der private Schlüssel.
- $p, q, \varphi(n)$ und d müssen geheim gehalten werden.

RSA: Beispiel einer Schlüsselgenerierung

- Wähle $p = 11$ und $q = 23$.
- Berechne das RSA-Modul $n = p \cdot q = 11 \cdot 23 = 253$.
- Wähle $e = 3$ (e ist zu $(p-1) \cdot (q-1) = 10 \cdot 22 = 2 \cdot 5 \cdot 2 \cdot 11$ teilerfremd und es gilt $1 < e < 220$).
- Das Paar $(253, 3)$ ist der öffentliche Schlüssel.
- Berechne das multiplikativ inverse Element d zu $e = 3$, d.h., es muss $d \cdot 3 \equiv 1 \pmod{220}$ gelten.
- d kann mithilfe des erweiterten Euklidischen Algorithmus berechnet werden.
- Es gilt $d = 147$ denn $147 \cdot 3 = 441 \equiv 1 \pmod{220}$.

RSA: Verschlüsseln einer Nachricht mit (n, e)

- Alice findet den öffentlichen Schlüssel (n, e) von Bob auf dessen Homepage.
- Sie verschlüsselt ihre Nachricht m (wobei m eine natürliche Zahl mit $0 \leq m < n$ ist) zu

$$c = m^e \bmod n$$

  und sendet c an Bob.
- e heißt deshalb Verschlüsselungsexponent.
- In unserem Beispiel verschlüsselt Alice ihre Nachricht 26 zu

$$m^e \bmod n = 26^3 \bmod 253 = 17576 \bmod 253 = 119$$

RSA: Entschlüsseln einer Nachricht c mit d und n

- Bob erhält die verschlüsselte Nachricht $c = 119$ von Alice.
- Er entschlüsselt den Geheimtext c, indem er

$$c^d \bmod n$$

  berechnet.
- d heißt deshalb Entschlüsselungsexponent.
- In unserem Beispiel muss Bob $119^d \bmod 253$ berechnen.
- Erinnerung: $c \equiv m^e \pmod{n}$.
- RSA beruht auf folgendem Theorem:

$$m \equiv (m^e)^d \pmod{n}$$

RSA: Korrektheit

Beh.: Im RSA-Kryptosystem gilt

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Beweis: Da $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$, gibt es ein $k \in \mathbb{Z}$ mit

$$m^{ed} = m^{1+k(p-1)(q-1)} = m \cdot m^{k(p-1)(q-1)}$$

Wir erhalten $m^{ed} \equiv m \cdot (m^{(p-1)})^{k(q-1)} \equiv m \pmod{p}$, denn wenn

- p kein Teiler von m ist, so folgt $m^{(p-1)} \equiv 1 \pmod{p}$ mit dem kleinen Satz von Fermat und damit

$$m^{ed} \equiv m \cdot (m^{(p-1)})^{k(q-1)} \equiv m \cdot 1^{k(q-1)} \equiv m \pmod{p}$$

- p ein Teiler von m ist, so folgt $m^{ed} \equiv 0 \equiv m \pmod{p}$.

RSA: Korrektheit

Beh.: Im RSA-Kryptosystem gilt

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Beweis (Fortsetzung): Es gilt

- $m^{ed} \equiv m \pmod{p}$ (s.o.)
- $m^{ed} \equiv m \pmod{q}$ (analog)

Nun folgt $m^{ed} \equiv m \pmod{n}$ aus der Tatsache, dass p und q teilerfremd sind. Dies sieht man wie folgt:

- $x \equiv y \pmod{p}$ bedeutet $p \mid (x - y)$.
- Analoges gilt für q.
- Es folgt $p \cdot q \mid (x - y)$, weil p und q teilerfremd sind.
- D.h. $x \equiv y \pmod{n}$, weil $n = p \cdot q$.

RSA-Kryptosystem

Damit das RSA-Kryptosystem in der Praxis funktioniert, müssen folgende algorithmische Probleme effizient gelöst werden:

1. Wähle zufällig zwei sehr große Primzahlen (mit diesem Problem beschäftigen wir uns später).
2. Berechne ein multiplikatives Inverses d zu e modulo $\varphi(n)$. d kann mithilfe des erweiterten Euklidischen Algorithmus in $O(k^2)$ Zeit berechnet werden, wobei $k =$ Bitlänge.
3. Gegeben a, b und n, berechne a^b mod n.

(1) und (2) erfolgen nur einmal pro Schlüsselpaar.
(3) Die Berechnung der modularen Exponentiation erfolgt bei jeder Ver- und Entschlüsselung. Der Aufwand ist $O(k^3)$, wie wir sofort sehen werden.

Berechnen der modularen Exponentiation $a^b \pmod{n}$

- Betrachten wir die Bit-Komplexität von $a^b \pmod{n}$.
- Hierbei seien a, b, n jeweils Zahlen mit höchstens $k+1$ Bits.
- Es ist nicht effizient, diese Funktion auf die Weise

$$\underbrace{a \cdot a \cdots a}_{b\text{-mal}} \bmod n$$

  zu berechnen.
- Dies ergäbe die Bit-Komplexität $O(k^2 2^k)$, weil eine Multiplikation/Division die Bit-Komplexität $O(k^2)$ hat.

Berechnen der modularen Exponentiation $a^b \pmod n$

- Es geht wesentlich besser, indem man die Zahl b zunächst in ihre Binärdarstellung entwickelt: $b = (b_k b_{k-1} \ldots b_1 b_0)_2$.
- Anhand der Binärdarstellung kann man durch fortgesetztes Quadrieren – und Multiplikation mit a, falls $b_i = 1$ – die Potenz berechnen (square-and-multiply).
- Dieser Algorithmus der binären Exponentiation wurde bereits vor ca. 2200 Jahren in Indien entdeckt.
- Der folgende Algorithmus berechnet $a^b \pmod n$ auf diese Art:

```
PROC modexp( a, b, n );
d := 1;
FOR i := k DOWNTO 0 DO
    d := (d * d) mod n;
    IF (b_i == 1) THEN d := (d * a) mod n;
RETURN d
```

Berechnen der modularen Exponentiation a^b mod n

Beispiel $a = 119$ und $b = 147 = (10010011)_2$

- $i = 7$: $d = 1^2 \cdot a$
- $i = 6$: $d = a^2$
- $i = 5$: $d = a^4$
- $i = 4$: $d = a^8 \cdot a$
- $i = 3$: $d = a^{18}$
- $i = 2$: $d = a^{36}$
- $i = 1$: $d = a^{72} \cdot a$
- $i = 0$: $d = a^{146} \cdot a = a^b$

Alle Rechnungen erfolgen natürlich modulo n.

Begründung der Korrektheit der Prozedur $modexp(a, b, n)$

- Angenommen wir haben bereits a^x korrekt berechnet, hierbei sei x die Binärdarstellung des Exponenten.
- Fügt man eine 0 hinter x an, dann gilt: $a^{x0} = (a^x)^2$.
- Das heißt man erhält die neue Zahl aus der alten durch Quadrieren.
- Fügt man eine 1 an, so gilt $a^{x1} = (a^x)^2 \cdot a$.
- Genau diese Rechenoperationen führt die Methode square-and-multiply aus.
- Jedes Zwischenergebnis kann dabei modulo n reduziert werden.

Bit-Komplexität von *modexp*

- Die Anzahl der Iterationen der Schleife ist gleich der Anzahl der Bits von b.
- Pro Iteration sind höchstens zwei Multiplikationen/Divisionen nötig.
- Eine Multiplikation/Division hat die Bit-Komplexität $O(k^2)$.
- Somit hat *modexp* die Bit-Komplexität $O(k^3)$.

Bem.: In der Praxis wird oft die Primzahl $e = 2^{16} + 1 = 65537$ als Verschlüsselungsexponent gewählt.

- In diesem Fall kommt das Verschlüsseln mit einer konstanten Anzahl (17) an modularen Multiplikationen aus.
- Also hat *modexp* hier die Bit-Komplexität $O(k^2)$.

Ausblick zur RSA-Kryptoanalyse

Bei der Kryptoanalyse des RSA-Verfahrens unterscheidet man zwischen zwei Problemen.

1. RSA-Problem (RSA im schwachen Sinne brechen):
   - Gegeben sind der öffentliche Schlüssel (n, e) sowie der Geheimtext c.
   - Gesucht wird der Klartext m, wobei $m^e \equiv c \pmod{n}$ gilt.
   - Das Problem liegt hier in der Schwierigkeit, e-te Wurzeln modulo n zu ziehen.
2. RSA-Schlüsselproblem (RSA im starken Sinne brechen):
   - Gegeben ist der öffentliche Schlüssel (n, e).
   - Gesucht wird der geheime Schlüssel d, wobei $d \cdot e \equiv 1 \pmod{\varphi(n)}$ gilt.
   - Hier liegt die Schwierigkeit darin, $\varphi(n) = (p-1)(q-1)$ ohne Kenntnis der Faktoren p und q zu berechnen.

Ausblick zur RSA-Kryptoanalyse

- Die Schwierigkeit der beiden Probleme beruht im Wesentlichen auf der Schwierigkeit des Faktorisierungsproblems.
- Man möchte $n = p \cdot q$ für sehr große Primzahlen p und q faktorisieren.
- Diese Primfaktorzerlegung ist für große Zahlen mit den heute bekannten Verfahren praktisch nicht durchführbar.
- Die Konstruktion eines leistungsfähigen Quantencomputers, der die Faktorisierung von Zahlen durch Verwendung des Shor-Algorithmus effizient durchführen könnte, könnte das RSA-Kryptosystem obsolet machen.

Untergruppen

- Eine Teilmenge U einer Gruppe G heißt Untergruppe von G, wenn U mit der Verknüpfung von G selbst eine Gruppe ist.
- Es sei k die Ordnung des Elements $a \in \mathbb{Z}_n^*$.
- Definiere $\langle a \rangle = \{a^1, a^2, \ldots, a^k = 1\}$.
- $\langle a \rangle$ ist eine kommutative Untergruppe von $\mathbb{Z}_n^*$:
  - $\langle a \rangle$ ist abgeschlossen unter Multiplikation, denn $a^i \cdot a^j \equiv a^{i+j} \equiv a^{(i+j) \bmod k} \pmod{n}$
  - es gilt das Assoziativgesetz
  - es gilt das Kommutativgesetz
  - es gibt ein neutrales Element, das ist die 1
  - jedes Element a^i hat ein Inverses, nämlich a^{k-i}

Zyklische Gruppen, Primitivwurzeln

- Eine Gruppe G heißt *zyklisch*, wenn es ein Element $a \in G$ gibt, so dass $\langle a \rangle = G$.
- Das Element a heißt dann *Erzeuger* der betreffenden Gruppe bzw. im Falle von $\mathbb{Z}_n^*$ eine *Primitivwurzel* modulo n.
- Beachte: nicht für jedes n ist die Gruppe $\mathbb{Z}_n^*$ zyklisch (z.B. ist $\mathbb{Z}_{15}^*$ nicht zyklisch).
- Beachte: für zyklisches $\mathbb{Z}_n^*$ ist nicht jedes $a \in \mathbb{Z}_n^*$ eine Primitivwurzel (Beispiel folgt unten).

Zyklische Gruppen: Beispiel

Betrachte $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Es gilt

$$\begin{aligned}
\langle 1 \rangle &= \{1\} \\
\langle 2 \rangle &= \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\} \\
\langle 3 \rangle &= \{1, 3, 9, 5, 4\} \\
\langle 4 \rangle &= \{1, 4, 5, 9, 3\} \\
\langle 5 \rangle &= \{1, 5, 3, 4, 9\} \\
\langle 6 \rangle &= \{1, 6, 3, 7, 9, 10, 5, 8, 4, 2\} \\
\langle 7 \rangle &= \{1, 7, 5, 2, 3, 10, 4, 6, 9, 8\} \\
\langle 8 \rangle &= \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\} \\
\langle 9 \rangle &= \{1, 9, 4, 3, 5\} \\
\langle 10 \rangle &= \{1, 10\}
\end{aligned}$$

Es gibt hier 4 Primitivwurzeln, und zwar 2, 6, 7, 8.

Zyklische Gruppen

Satz von Gauß (ohne Beweis): Die Gruppe $\mathbb{Z}_n^*$ ist zyklisch gdw. n eine der folgenden Formen hat:

$$n = 1, 2, 4, p^k, 2p^k$$

wobei p eine ungerade Primzahl ist und $k \in \mathbb{N}$.

- Beispielsweise ist $\mathbb{Z}_{18}^*$ zyklisch, da $18 = 2 \cdot 3^2$.
- Dagegen ist $\mathbb{Z}_{15}^*$ nicht zyklisch, weil $15 = 3 \cdot 5$.
- Insbesondere ist $\mathbb{Z}_n^*$ immer zyklisch, wenn n eine Primzahl ist (nur diesen Fall benötigen wir im Folgenden).
- Diesen Primzahl-Fall werden wir auch beweisen.

Zyklische Gruppen: Anzahl der Erzeuger

Lemma: Ist $k \in \mathbb{N}$ die Ordnung von $a \in G$ und $\ell \in \mathbb{N}$, so ist $\frac{k}{ggT(k,\ell)}$ die Ordnung von a^ℓ.

Beweis:

- Es gilt $(a^\ell)^{k/ggT(k,\ell)} = (a^k)^{\ell/ggT(k,\ell)} = 1^{\ell/ggT(k,\ell)} = 1$.
- Also ist $\frac{k}{ggT(k,\ell)}$ ein Vielfaches der Ordnung von a^ℓ.
- Wir zeigen, dass $\frac{k}{ggT(k,\ell)}$ die Ordnung m von a^ℓ teilt.
- (Da $\frac{k}{ggT(k,\ell)}$ ein Vielfaches von m und Teiler von m ist, folgt dann $m = \frac{k}{ggT(k,\ell)}$.)
- Es muss k ein Teiler von $\ell \cdot m$ sein, weil $a^{\ell \cdot m} = (a^\ell)^m = 1$.
- Dann ist auch $\frac{k}{ggT(k,\ell)}$ ein Teiler von $\frac{\ell \cdot m}{ggT(k,\ell)}$.
- Es gilt $\mathrm{ggT}(\frac{k}{ggT(k,\ell)}, \frac{\ell}{ggT(k,\ell)}) = 1$. Da $\frac{k}{ggT(k,\ell)}$ kein Teiler von $\frac{\ell}{ggT(k,\ell)}$ ist, muss es ein Teiler von m sein.

Zyklische Gruppen: Anzahl der Erzeuger

Theorem: Ist die endliche Gruppe G zyklisch, so hat G genau $\varphi(|G|)$ Erzeuger.

Beweis:

- Ein Element aus G erzeugt die Gruppe G gdw. es die Ordnung $|G|$ hat.
- Wir bestimmen die Anzahl der Elemente der Ordnung $|G|$.
- Sei a ein Erzeuger von G (a existiert, weil G zyklisch ist).
- Dann ist $G = \{a^\ell \mid 0 \leq \ell < |G|\} = \{a^\ell \mid 1 \leq \ell \leq |G|\}$.
- Ein Element dieser Menge hat die Ordnung $|G|$ gdw. $\mathrm{ggT}(|G|, \ell) = 1$ (wurde im vorherigen Lemma bewiesen).
- Also ist die Anzahl der Erzeuger von G genau $\varphi(|G|)$.

Zyklische Gruppen: Anzahl der Erzeuger

Korollar: Die Anzahl der Primitivwurzeln einer zyklischen Gruppe $\mathbb{Z}_n^*$ beträgt $\varphi(\varphi(n))$.

Beweis:

- Laut obigem Theorem hat $\mathbb{Z}_n^*$ genau $\varphi(|\mathbb{Z}_n^*|)$ Erzeuger.
- Es gilt $|\mathbb{Z}_n^*| = \varphi(n)$.

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Zum Beweis, dass $\mathbb{Z}_n^*$ zyklisch ist, sofern n Primzahl ist, benötigen wir folgendes Lemma für den Fall $m = n - 1$.

Lemma: Die Summe aller $\varphi(d)$-Werte ergibt m, wenn man alle positiven Teiler d von m durchläuft. Symbolisch:

$$\sum_{d|m, d>0} \varphi(d) = m$$

Beispiel: Die Teiler von $m = 10$ sind 1, 2, 5, 10.
Es gilt

$$\varphi(1) + \varphi(2) + \varphi(5) + \varphi(10) = 1 + 1 + 4 + 4 = 10$$

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Lemma:

$$\sum_{d|m, d>0} \varphi(d) = m$$

Beweis: Im Folgenden betrachten wir nur positive Teiler d von m.

- Definiere für jeden Teiler d von m die Menge

$$A_d(m) = \{x \in \mathbb{Z}_m \mid ggT(x, m) = d\}$$

- Die Mengen $A_d(m)$ sind paarweise disjunkt und deren Vereinigung (über alle Teiler d von m) ergibt $\mathbb{Z}_m$.
- Also folgt $\sum_{d|m} |A_d(m)| = m$.
- Ferner gilt $|A_d(m)| = \varphi(m/d)$, denn

$$\varphi(m/d) = |\{y \mid 1 \leq y \leq m/d,\, ggT(m/d, y) = 1\}|$$

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Lemma:

$$\sum_{d|m, d>0} \varphi(d) = m$$

Beweis (Fortsetzung):

- Es gilt

$$\sum_{d|m} \varphi(d) = \sum_{d|m} \varphi(m/d)$$

  da die Menge der positiven Teiler von m genau die Menge $\{m/d : d|m, d > 0\}$ ist, also insgesamt über dieselbe Menge von Werten aufsummiert wird.

- Insgesamt folgt

$$\sum_{d|m} \varphi(d) = \sum_{d|m} \varphi(m/d) = \sum_{d|m} |A_d(m)| = m$$

Zyklische Gruppen: Beispiel

Betrachte $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Es gilt

$$\begin{aligned}
\langle 1 \rangle &= \{1\} \\
\langle 2 \rangle &= \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\} \\
\langle 3 \rangle &= \{1, 3, 9, 5, 4\} \\
\langle 4 \rangle &= \{1, 4, 5, 9, 3\} \\
\langle 5 \rangle &= \{1, 5, 3, 4, 9\} \\
\langle 6 \rangle &= \{1, 6, 3, 7, 9, 10, 5, 8, 4, 2\} \\
\langle 7 \rangle &= \{1, 7, 5, 2, 3, 10, 4, 6, 9, 8\} \\
\langle 8 \rangle &= \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\} \\
\langle 9 \rangle &= \{1, 9, 4, 3, 5\} \\
\langle 10 \rangle &= \{1, 10\}
\end{aligned}$$

Es gibt hier 4 Primitivwurzeln, und zwar 2, 6, 7, 8.
Bem.: $\varphi(\varphi(11)) = \varphi(11 - 1) = \varphi(2) \cdot \varphi(5) = 1 \cdot 4$

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Theorem: Wenn n Primzahl ist, dann ist $\mathbb{Z}_n^*$ zyklisch.
Beweis:

- Definiere $B_d(n) = \{x \in \mathbb{Z}_n^* \mid$ die Ordnung von x ist $d\}$.
- Dann gilt $\sum_{d|n-1} |B_d(n)| = n - 1$, weil $\varphi(n) = n - 1$ und d ein Teiler von $\varphi(n)$ sein muss.
- Wir zeigen zunächst, dass aus $|B_d(n)| > 0$ die Gleichheit $|B_d(n)| = \varphi(d)$ folgt.
- Sei $a \in B_d(n)$. Die Potenzen a^ℓ, $0 \leq \ell < d$, sind paarweise verschieden und alle Nullstellen des Polynoms $x^d - 1$, weil

$$(a^\ell)^d - 1 = (a^d)^\ell - 1 = 1^\ell - 1 = 0$$

- In dem Körper $(\mathbb{Z}_n, +, *)$ hat $x^d - 1$ höchstens d Nullstellen.
- Also hat das Polynom $x^d - 1$ die d Nullstellen $a^0, \ldots, a^{d-1}$.
- Nun ist aber jedes Element von $(\mathbb{Z}_n, +, *)$ der Ordnung d eine Nullstelle des Polynoms $x^d - 1$ und daher eine Potenz von a.

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Theorem: Wenn n Primzahl ist, dann ist $\mathbb{Z}_n^*$ zyklisch.

Beweis (Fortsetzung):

- Erinnerung: Ist $d \in \mathbb{N}$ die Ordnung von $a \in \mathbb{Z}_n^*$ und $\ell \in \mathbb{N}$, so ist d die Ordnung von a^ℓ gdw. $ggT(d, \ell) = 1$.
- Es folgt

$$\begin{aligned} |B_d(n)| &= |\{a^\ell \mid 0 \leq \ell < d \text{ und die Ordnung von } a^\ell \text{ ist } d\}| \\ &= |\{a^\ell \mid 0 \leq \ell < d \text{ und } ggT(d, \ell) = 1\}| \\ &= |\{\ell \mid 0 \leq \ell < d \text{ und } ggT(d, \ell) = 1\}| \\ &= |\{\ell \mid 1 \leq \ell \leq d \text{ und } ggT(d, \ell) = 1\}| \end{aligned}$$

- Also haben wir gezeigt, aus $|B_d(n)| > 0$ die Gleichheit $|B_d(n)| = \varphi(d)$ folgt.

$\mathbb{Z}_n^*$ ist zyklisch falls n Primzahl

Theorem: Wenn n Primzahl ist, dann ist $\mathbb{Z}_n^*$ zyklisch.

Beweis (Fortsetzung): Nun zeigen wir, dass $|B_d(n)| > 0$ für jeden Teiler d von $n-1$ gilt.

- Wenn es einen Teiler d von $n-1$ mit $|B_d(n)| = 0$ geben würde, so würde folgen

$$n - 1 = \sum_{d|n-1} |B_d(n)| < \sum_{d|n-1} \varphi(d)$$

- Dies ergibt ein Widerspruch zum oben hergeleiteten $\sum_{d|n-1} \varphi(d) = n - 1$.
- Es gilt also $|B_d(n)| > 0$ für jeden Teiler d von $n-1$.
- Insbesondere ist deshalb $B_{n-1}(n)$ nicht leer (und hat damit $\varphi(n-1)$ viele Elemente).
- Diese sind die Primitivwurzeln modulo n.
- Also ist $\mathbb{Z}_n^*$ zyklisch.

Primitivwurzeln finden

- Erinnerung: Die Anzahl der Primitivwurzeln einer zyklischen Gruppe $\mathbb{Z}_n^*$ beträgt $\varphi(\varphi(n))$.
- Es gibt also $\varphi(n) - \varphi(\varphi(n))$ viele Elemente in $\mathbb{Z}_n^*$, die keine Primitivwurzeln sind.
- Wie findet man eine Primitivwurzel?
- Probabilistischer Algorithmus für den Fall, dass die Primfaktorzerlegung von $\varphi(n)$ bekannt ist: Man wählt $a \in \mathbb{Z}_n^*$ (mehrfach) zufällig und testet mit dem folgenden Kriterium, ob a eine Primitivwurzel modulo n ist.

Primitivwurzel-Kriterium

Satz: Falls $\mathbb{Z}_n^*$ zyklisch ist, so ist $a \in \mathbb{Z}_n^*$ genau dann eine Primitivwurzel modulo n, wenn $a^{\varphi(n)/q} \neq 1$ für alle Primzahlen q, die $\varphi(n)$ teilen.

Beweis: Die Folge der Potenzen $a^1, a^2, \ldots$ wiederholt sich periodisch und nach dem Satz von Euler gilt $a^{\varphi(n)} = 1$.

- Wenn a eine Primitivwurzel ist, so ist $\varphi(n)$ die Ordnung von a und die Behauptung folgt.
- Wenn a keine Primitivwurzel ist, so gilt für die Ordnung k von a, dass $k < \varphi(n)$ und k muss ein Teiler von $\varphi(n)$ sein.
- D.h. $k \cdot \ell = \varphi(n)$ für ein $\ell \in \mathbb{N}$ mit $\ell \geq 2$.
- Sei q ein Primfaktor von ℓ. Dann gilt $\ell = q \cdot m$ für ein $m \in \mathbb{N}$ und $k \cdot m = \varphi(n)/q$.
- Nun folgt die Behauptung aus

$$a^{\varphi(n)/q} = a^{k \cdot m} = (a^k)^m = 1^m = 1$$

Primitivwurzel-Kriterium

- Sei nun n eine Primzahl, also $\varphi(n) = n - 1$.
- Das Primitivwurzel-Kriterium stellt einen effizienten Test dar, wenn die Primfaktoren von $n - 1$ bekannt sind.
- Man wählt a mehrfach zufällig, und führt den Primitivwurzeltest durch.
- Schwierig ist es jedoch, die Primitivwurzeleigenschaft festzustellen, wenn die Faktorisierung von $n - 1$ nicht bekannt ist.

Primitivwurzel-Kriterium

- Bei manchen Krypto-Verfahren (Diffie-Hellman, ElGamal) muss initial eine große Zufallsprimzahl n gewählt werden und ebenso eine Primitivwurzel, also eine Zahl a, die das Primitivwurzel-Kriterium erfüllt.
- Man geht hier so vor, zuerst eine Primzahl q zu finden und dann zu testen, ob $n = 2q + 1$ auch Primzahl ist.
- Im Erfolgsfall nennt man dann n eine *sichere Primzahl* und q eine *Sophie Germain-Primzahl*.
- Nun hat man die Faktorisierung $n - 1 = 2 \cdot q$ für das Primitivwurzel-Kriterium zur Verfügung.
- Da $\varphi(n-1) = \varphi(2q) = q - 1 = (n-3)/2$ findet man eine Primitivwurzel nach wenigen (nämlich ca. 2) Zufallsversuchen.

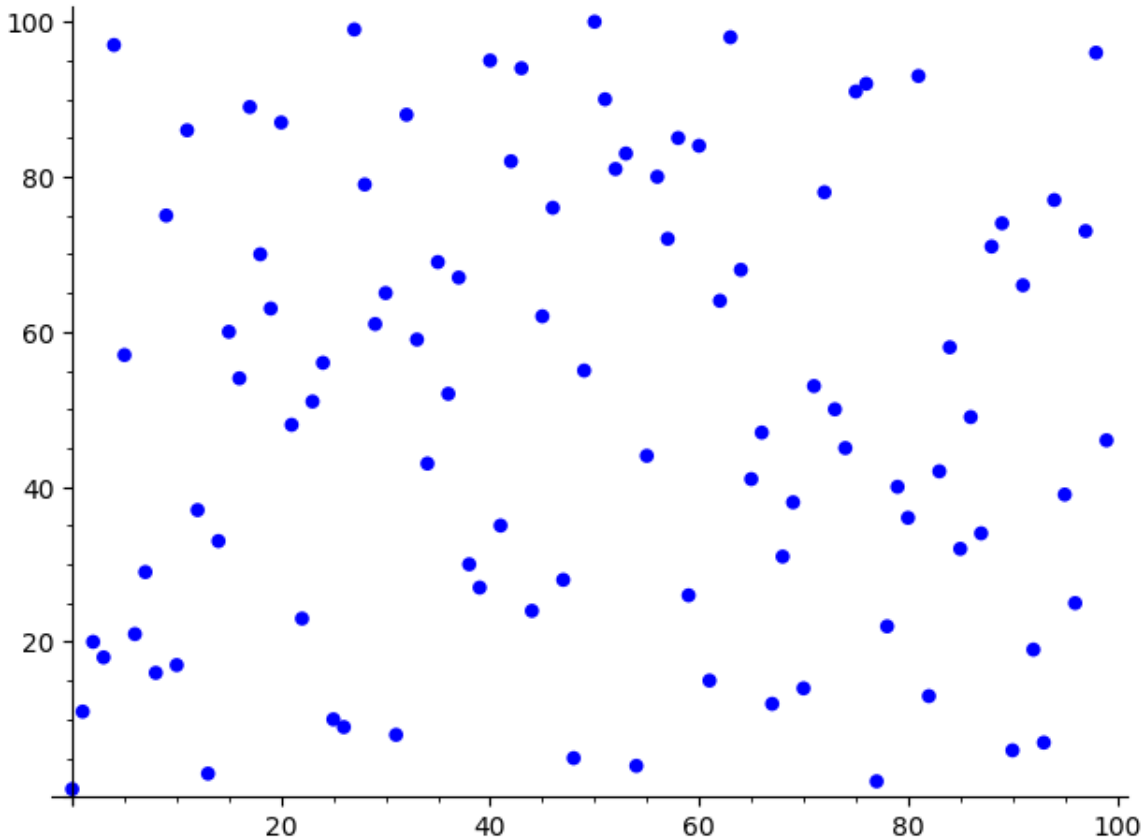
Primitivwurzel-Kriterium: Beispiel

- $n = 227$ ist eine sichere Primzahl, denn $(n-1)/2 = 113$ ist ebenfalls eine Primzahl.
- Gemäß dem Primitivwurzel-Kriterium ist eine Zahl $a < n$ eine Primitivwurzel gdw. sowohl a^2 als auch a^{113} nicht kongruent 1 modulo 227 sind.
- Zum Beispiel ist $a = 5$ eine Primitivwurzel.
- Es gibt $\varphi(226) = \varphi(2 \cdot 113) = 112$ Primitivwurzeln.

Exponentiation und diskreter Logarithmus

Sei $\mathbb{Z}_n^*$ eine zyklische Gruppe und sei a eine Primitivwurzel modulo n, wobei wir uns auf n Primzahl beschränken.

Ein Beispiel: $\mathbb{Z}_{101}^*$ ist zyklisch (da 101 Primzahl) und $a = 11$ ist eine Primitivwurzel. Das folgende Diagramm zeigt den Funktionsverlauf von a^i mod 101 für $i = 0, 1, \ldots, 100$.



Eponentiation und diskreter Logarithmus

Sei $\mathbb{Z}_n^*$ eine zyklische Gruppe und sei a eine Primitivwurzel modulo n, wobei wir uns auf n Primzahl beschränken.

- Dann hat jede Zahl $x \in \mathbb{Z}_n^*$ einen eindeutigen Exponenten $k \in \{0, 1, 2, \ldots, n-2\}$ (auch *Index* genannt) mit $x = a^k$.
- Dieser Exponent k ist der *diskrete Logarithmus* von x (zur Basis a, modulo n).
- Das Berechnen des Diskreten Logarithmus ist eines der schwierigen Probleme, für das nur exponentielle Algorithmen bekannt sind.
- Auf dieser Tatsache beruht die Sicherheit von mehreren kryptographischen Systemen.
- Andererseits ist das Berechnen von a^k mod n effizient möglich (mittels der Prozedur modexp).
- Daher geht man davon aus, dass diese modulare Exponentiation eine für die Kryptographie wichtige Einwegfunktion ist (mehr dazu später).

Exponentiation und diskreter Logarithmus

Beispiel: Betrachte $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Es gilt

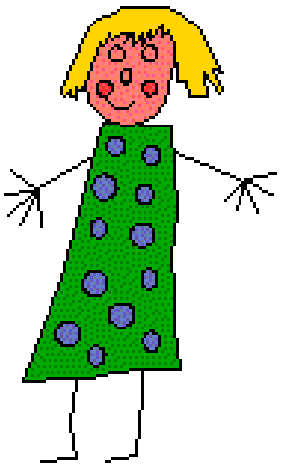
$$\begin{aligned}
\langle 1 \rangle &= \{1\} \\
\langle 2 \rangle &= \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\} \\
\langle 3 \rangle &= \{1, 3, 9, 5, 4\} \\
\langle 4 \rangle &= \{1, 4, 5, 9, 3\} \\
\langle 5 \rangle &= \{1, 5, 3, 4, 9\} \\
\langle 6 \rangle &= \{1, 6, 3, 7, 9, 10, 5, 8, 4, 2\} \\
\langle 7 \rangle &= \{1, 7, 5, 2, 3, 10, 4, 6, 9, 8\} \\
\langle 8 \rangle &= \{1, 8, 9, 6, 4, 10, 3, 2, 5, 7\} \\
\langle 9 \rangle &= \{1, 9, 4, 3, 5\} \\
\langle 10 \rangle &= \{1, 10\}
\end{aligned}$$

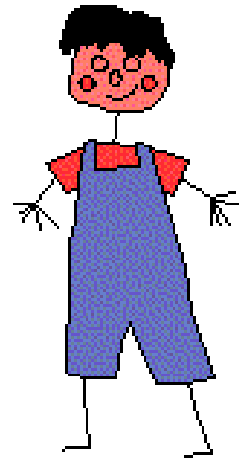
Aus obiger Liste ergeben sich die diskreten Logarithmen von $1, 2, \ldots, 10$ (zur Basis 7 und modulo 11) als $0, 3, 4, 6, 2, 7, 1, 9, 8, 5$.

Diffie-Hellman-Schlüsselvereinbarung

- Initial wird eine große Primzahl n und eine Primitivwurzel a modulo n festgelegt.
- Diese Zahlen sind öffentlich und feste Parameter des Kryptosystems.
- Nun wählen Alice und Bob je eine Zufallszahl kleiner n.
- Alice wählt x und Bob wählt y.
- Alice berechnet $\tilde{x} = (a^x \bmod n)$ und schickt $\tilde{x}$ an Bob.
- Bob berechnet $\tilde{y} = (a^y \bmod n)$ und schickt $\tilde{y}$ an Alice.
- Nun berechnet Alice $z = (\tilde{y}^x \bmod n)$.
- Analog berechnet Bob $z = (\tilde{x}^y \bmod n)$.
- Beide haben jetzt die Zahl $z = (a^{xy} \bmod n)$ zur Verfügung.
- Nun kann z als Schlüssel für ein effizientes klassisches und als sicher erachtetes Kryptoverfahren verwendet werden, um die eigentliche Nachricht m zu übertragen.

Diffie-Hellman-Schlüsselvereinbarung



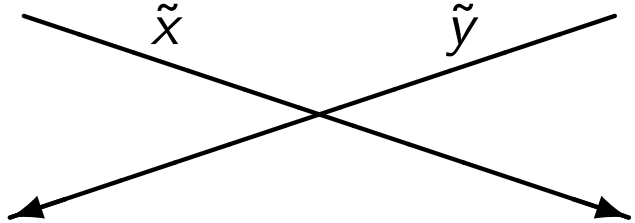


$x \in_R \mathbb{Z}_n^*$

$\text{modexp}(a, x, n) = \tilde{x}$

$y \in_R \mathbb{Z}_n^*$

$\tilde{y} = \text{modexp}(a, y, n)$



$\text{modexp}(\tilde{y}, x, n) = z$

$\text{modexp}(\tilde{x}, y, n) = z$

$E(m, z) = c$

$D(c, z) = m$

Diffie-Hellman-Schlüsselvereinbarung: Beispiel

- Wähle $n = 23$ (sichere Primzahl) und $a = 5$ (Primitivwurzel).
- Auflistung der Paare $(x, (5^x \bmod 23))$:
  $(0,1), (1,5), (2,2), (3,10), (4,4), (5,20), (6,8), (7,17),$
  $(8,16), (9,11), (10,9), (11,22), (12,18), (13,21), (14,13),$
  $(15,19), (16,3), (17,15), (18,6), (19,7), (20,12), (21,14), (22,1)$
- Nun wählen Alice und Bob Zufallszahlen.
- Alice wählt $x = 6$ und erhält damit $\tilde{x} = 8$.
- Bob wählt $y = 15$ und erhält damit $\tilde{y} = 19$.
- Alice sendet $\tilde{x}$ an Bob, und Bob sendet $\tilde{y}$ an Alice.
- Alice berechnet $z = (\tilde{y}^x \bmod n) = (19^6 \bmod 23) = 2$.
- Bob berechnet $z = (\tilde{x}^y \bmod n) = (8^{15} \bmod 23) = 2$.
- Nun kann die Nachricht m mittels des Schlüssels z klassisch verschlüsselt und an Bob gesandt werden. Dieser kann dann entschlüsseln, da er ebenfalls den Schlüssel z besitzt.

Zur Diffie-Hellman-Kryptoanalyse

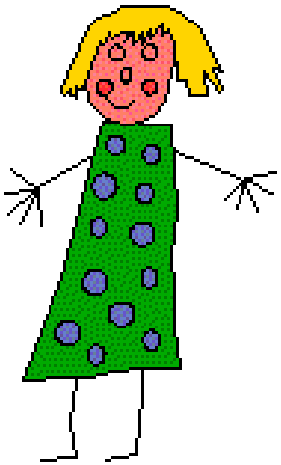
Das Diffie-Hellman-Problem beschreibt das Problem der Kryptoanalyse dieses Krypto-Systems:

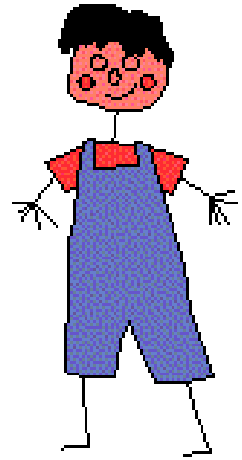
- Gegeben sei $(n, a, \tilde{x}, \tilde{y})$, wobei $\tilde{x} = a^x \bmod n$ und $\tilde{y} = a^x \bmod n$.
- Bestimme $z = a^{xy} \bmod n$.
- Das Diffie-Hellman-Problem ließe sich wie folgt lösen, wenn man den Diskreten Logarithmus effizient berechnen könnte.
- Man bestimmt den diskreten Logarithmus modulo n von $\tilde{x}$ und $\tilde{y}$ zur Basis a und kann dann damit z bestimmen.

No Key-System nach Shamir

- Initial wird eine große Primzahl n festgelegt.
- Alice möchte eine Nachricht $m < n$ an Bob schicken.
- Dazu wählt Alice eine Zufallszahl $a \in \mathbb{Z}_{n-1}^*$ und berechnet das zugehörige Inverse a' mit $aa' \equiv 1 \pmod{n-1}$.
- Sie schickt nun $c = (m^a \bmod n)$ an Bob.
- Dieser wählt ebenso eine Zufallszahl b und berechnet das Inverse b' modulo $n-1$.
- Nun schickt Bob die Zahl $c' = (c^b \bmod n)$ zurück an Alice.
- Alice entfernt ihre Verschlüsselung dadurch wieder, dass sie die empfangene Zahl mit a' potenziert (modulo n), also $c'' = (c')^{a'} = m^{aba'} = m^b \pmod{n}$.
- Wenn Bob diese Information zurück erhält, kann er durch Potenzieren mit b' entschlüsseln und erhält dadurch die Nachricht m.

No Key-System nach Shamir (alle Rechnungen modulo n)





m

$c = m^a$ →

$c' = c^b = m^{ab}$ ←

$c'' = (c')^{a'} = m^{aba'} = m^b$ →

$(c'')^{b'} = m^{bb'} = m$

No Key-System nach Shamir

Wir wollen noch die Grundlage des No Key-Systems beweisen.

Behauptung: Es gilt $m^{aa'} \equiv m \pmod{n}$.

Beweis: Da $aa' \equiv 1 \pmod{n-1}$ existiert ein k mit

$$m^{aa'} = m^{1+k(n-1)} = m \cdot (m^{(n-1)})^k \equiv m \cdot 1^k \pmod{n}$$

Beachte: Die Kongruenz $m^{(n-1)} \equiv 1 \pmod{n}$ gilt mit dem kleinen Satz von Fermat.

No Key-System nach Shamir

- Man kann die Methode, auch 3-Phasen-Protokoll genannt, mit der Metapher einer mit mehreren Schlössern verschließbaren Box erklären.
- Alice legt ihre Nachricht in die Box, verschließt sie mit ihrem Schloss und sendet sie an Bob.
- Bob hängt zusätzlich sein eigenes Schloss dran und schickt die Box zurück an Alice.
- Diese entfernt ihr Schloss und schickt die Box mit Bobs Schloss wieder auf die Reise.
- Nun kann Bob die Box öffnen.

No Key-System nach Shamir

- Da die Nachricht (die viele Blöcke lang sein kann) hier dreimal hin und her geschickt wird, erscheint diese Methode unangemessen aufwändig.
- Allerdings könnte man die Methode auch in hybrider Weise verwenden: man tauscht einen von Alice erzeugten Zufallsschlüssel in der beschriebenen Weise aus, bis Bob ebenfalls diesen Schlüssel besitzt.
- Anschließend wird erst die eigentliche Nachricht in klassischer Manier mit dem nun Alice und Bob bekannten Schlüssel verschickt.

No Key-System nach Shamir: Beispiel

- Die Primzahl sei $n = 23$.
- Alice wählt per Zufall die Zahl $a = 3$ (teilerfremd zu $n - 1 = 22$) und bestimmt mittels extggT das multiplikative Inverse $a' = 15$, modulo $n - 1 = 22$.
- Bob wählt $b = 9$ und bestimmt $b' = 5$.
- Die Nachricht $m = 10$ wird von Alice verschlüsselt zu $c = m^a = 10^3 = 11 \pmod{23}$ und an Bob geschickt.
- Bob verschlüsselt nochmals und schickt dann $c' = c^b = 11^9 = 19 \pmod{23}$ zurück an Alice.
- Diese verschlüsselt zu $c'' = (c')^{a'} = 19^{15} = 20 \pmod{23}$.
- Tatsächlich bedeutet dies, dass dadurch Alice ihre ursprüngliche Verschlüsselung entfernt hat, und nur Bobs Verschlüsselung verbleibt, denn $20 = m^b = 10^9 \pmod{23}$.
- Sobald Bob diese Nachricht erhält, kann er entschlüsseln: $(c'')^{b'} = m^{bb'} = 20^5 = 10 = m \pmod{23}$.

Zur Kryptoanalyse des No Key-Systems

Wenn es möglich wäre, den Diskreten Logarithmus bzgl. eines Erzeugers g der zyklischen Gruppe $\mathbb{Z}_n^*$ effizient zu bestimmen, so könnte man das No Key-System wie folgt brechen:

- Berechne den Diskreten Logarithmus von $m^a = g^i$ sowie $m^{ab} = g^j$, also i und j.
- Erinnerung: Es sei k die Ordnung des Elements $g \in \mathbb{Z}_n^*$ und $\ell, m \in \mathbb{Z}$. Dann gilt

$$g^\ell \equiv g^m \pmod{n} \text{ gdw. } \ell \equiv m \pmod{k}$$

- Hier: $g^j \equiv g^{ib} \pmod{n}$ gdw. $j \equiv ib \pmod{n-1}$, weil die Ordnung von g gerade $\varphi(n) = n - 1$ ist.
- Solcherart kann man b und damit auch b' ermitteln.
- Man erhält nun m durch die Berechnung von $m^{bb'}$.

ElGamal-Verfahren

- Dieses Public Key-Verfahren hat große Ähnlichkeit mit dem Diffie-Hellman-Verfahren.
- Erinnerung: Dort sind Alice und Bob gleichberechtigt und im Prinzip gleichzeitig an der Herstellung der Zufallszahl z beteiligt (die anschließend als klassischer Schlüssel verwendet wird).
- Im Unterschied dazu wird bei ElGamal die Herstellung von y und $\tilde{y}$ auf der Seite von Bob (als Nachrichtenempfänger) in die Initialisierungsphase verschoben.
- Alice stellt erst im Ablauf des Protokolls x und $\tilde{x}$ her (als Einmal-Schlüssel, session key), wenn sie eine Nachricht an Bob senden möchte.

ElGamal-Verfahren

- Wie bei Diffie-Hellman wird eine große Primzahl n und eine Primitivwurzel a modulo n festgelegt.
- Initial wählt der spätere Nachrichtenempfänger (Bob) eine Zufallszahl $y < n$ und berechnet $\tilde{y} = (a^y \bmod n)$.
- Nun ist $\tilde{y}$ der öffentliche Schlüssel und y der geheime Schlüssel.
- Alice möchte an Bob die Nachricht $m < n$ schicken.
- Sie kennt die öffentlichen Informationen n, a und $\tilde{y}$.
- Alice wählt eine Zufallszahl $x < n$ und berechnet $\tilde{x} = (a^x \bmod n)$.
- Dann berechnet Alice $((\tilde{y})^x \bmod n) = z$.
- Dies ergibt wie bei Diffie-Hellman die Zufallszahl z.

ElGamal-Verfahren (Fortsetzung)

- Mit Hilfe von z wird (wie bei klassischer Kryptographie) die Nachricht m verschlüsselt zu $((m \cdot z) \bmod n) = c$.
- Alice sendet an Bob die Information $(\tilde{x}, c)$.
- Damit kann Bob nun auch mittels $((\tilde{x})^y \bmod n) = z$ die Zufallszahl z erfahren, denn $((\tilde{x})^y = a^{xy} = (\tilde{y})^x \bmod n) = z$.
- Er entschlüsselt dann mittels $((c \cdot z^{-1}) \bmod n) = m$.
- Um das Inverse z^{-1} zu berechnen, das die Verschlüsselung wieder aufhebt, muss Bob einen Aufruf von $\mathrm{extggT}(z, n)$ durchführen.
- Bemerkung: Alternativ könnten Alice und Bob eine noch einfachere Ver- und Entschlüsselung durchführen, indem m bzw. c mit z bitweise mittels XOR verküpft wird, also $c = m \oplus z$ bzw. $m = c \oplus z$

ElGamal-Verfahren: Vergleich mit RSA

- Bei RSA wird auf deterministiche Weise verschlüsselt (aus demselben Klartext entsteht immer dieselbe Chiffre).
- Dagegen ist ElGamal probabilistisch: abhängig von der Zufallszahl z entstehen verschiedene mögliche Chiffren.
- Dieser zusätzliche Sicherheitsaspekt wird erkauft dadurch, dass die zu übertragende Information doppelt so lang ist wie bei RSA.

ElGamal-Verfahren: Beispiel

- Die Primzahl sei $n = 23$ und die Primitivwurzel sei $a = 5$.
- Bob wählt die Zufallszahl $y = 15$ und berechnet $\tilde{y} = 19$.
- Öffentlich sind nun n, a und $\tilde{y}$.
- Der geheime Teil von Bobs Schlüssel ist y.
- Alice möchte Bob die Nachricht $m = 10$ senden.
- Dazu wählt sie die Zufallszahl $x = 6$ und berechnet $\tilde{x} = 8$.
- Durch $((\tilde{y})^x \bmod n)$ ergibt sich der Schlüssel $z = 2$.
- Mit diesem Schlüssel wird die Nachricht per modularer Multiplikation verschlüsselt: $c = ((m \cdot z) \bmod n) = 20$.
- Nun sendet Alice das Paar $(\tilde{x}, c) = (8, 20)$ an Bob.
- Damit berechnet Bob zunächst $z = ((\tilde{x})^y \bmod n) = (8^{15} \bmod 23) = 2$.
- Um Alice's Verschlüsselung rückgängig zu machen, berechnet er nun $(z^{-1} \bmod n) = 12$ (mittels extggT).
- Er erhält: $((c \cdot z^{-1}) \bmod n) = ((20 \cdot 12) \bmod 23) = 10 = m$.

Zur Kryptoanalyse des ElGamal-Verfahrens

Das ElGamal-Verfahren ließe sich knacken, wenn man die Funktion

$$(n, a, \tilde{y}, \tilde{x}) \mapsto z, \text{ wobei } z = a^{xy} \bmod n$$

effizient berechnen könnte.

Dies entspricht genau dem Diffie-Hellman-Problem.

Erinnerung: Das Diffie-Hellman-Problem ließe sich lösen, wenn man den Diskreten Logarithmus effizient berechnen könnte.

Quadratische Reste

Definition: Ein quadratischer Rest modulo n ist eine Zahl $a \in \mathbb{Z}_n^*$, so dass ein $b \in \mathbb{Z}_n^*$ existiert mit $b^2 \equiv a \pmod{n}$. Ist a ein quadratischer Rest modulo n, so nennen wir b eine Quadratwurzel von a modulo n.

- Beispiel: Wegen $5^2 \equiv 3 \pmod{11}$ ist 3 ein quadratischer Rest modulo 11 und 5 eine Quadratwurzel von 3 modulo 11.

Sei $\mathbb{Z}_n^*$ zyklisch und $g \in \mathbb{Z}_n^*$ eine Primitivwurzel.

- Wenn k eine gerade Zahl ist, so ist $a = g^k$ ist ein quadratischer Rest modulo n.
- Wähle als Quadratwurzel $b = g^{k/2}$.

Quadratische Reste

Satz: Sei n eine ungerade Primzahl. Wenn $a \in \mathbb{Z}_n^*$ ein quadratischer Rest modulo n ist, dann besitzt die Kongruenzgleichung $x^2 \equiv a \pmod{n}$ genau zwei Lösungen.

Beweis:

- Nach Definition existiert ein $b \in \mathbb{Z}_n^*$ mit $b^2 \equiv a \pmod{n}$.
- Beh.: $n - b$ ist ebenfalls eine Lösung. Wir rechnen nach:

$$(n-b)^2 \equiv (n^2 - 2bn + b^2) \equiv b^2 \equiv a \pmod{n}$$

- Die beiden Lösungen b und $n - b$ sind verschieden, da aus $n - b = b$ der Widerspruch $n = 2b$ folgen würde.
- Weitere Lösungen gibt es nicht, weil das Polynom $x^2 - a \in \mathbb{Z}_n[x]$ höchstens zwei Nullstellen hat.

Quadratische Reste

Korollar: Modulo einer Primzahl $n > 2$ gibt es genau $(n-1)/2$ viele quadratische Reste und ebenso viele quadratische Nichtreste.

Beweis: Die folgenden Zahlen sind quadratische Reste:

$$1^2 \mod n,\ 2^2 \mod n, \ldots, (\frac{n-1}{2})^2 \mod n$$

- Sie sind paarweise verschieden, weil für jede Zahl $b \in \{1, \ldots, \frac{n-1}{2}\}$ die Kongruenzgleichung $b^2 \equiv a \pmod{n}$ genau die zwei Lösungen b und $n - b$ hat.
- Da die oben genannten Quadratwurzeln auch paarweise verschieden sind (es also $n-1$ viele gibt), existieren keine weiteren quadratischen Reste.
- Alle quadratischen Reste kommen also in der obigen Folge vor.

Euler-Kriterium

Satz: Sei $n > 2$ eine Primzahl. Dann ist $a \in \mathbb{Z}_n^*$ ein quadratischer Rest modulo n genau dann, wenn $a^{(n-1)/2} \equiv 1 \pmod{n}$.

Beweis: "$\Rightarrow$" Wenn eine Quadratwurzel b von a existiert, so folgt: $a^{(n-1)/2} \equiv (b^2)^{(n-1)/2} \equiv b^{n-1} \equiv 1 \pmod{n}$ (Satz von Fermat).

"$\Leftarrow$" Umgekehrt gelte $a^{(n-1)/2} \equiv 1 \pmod{n}$.

- Für eine Primitivwurzel g existiert ein $k \in \mathbb{N}$ mit $a = g^k$.
- Es folgt $g^{k(n-1)/2} \equiv 1 \equiv g^0 \pmod{n}$.
- $g^{k(n-1)/2} \equiv g^0 \pmod{n}$ gdw. $k(n-1)/2 \equiv 0 \pmod{n-1}$ impliziert, dass $k/2$ eine ganze Zahl und damit k eine gerade Zahl ist.
- Also ist a ein quadratischer Rest.

Quadratwurzeln berechnen

Sei $n > 2$ eine Primzahl und $a \in \mathbb{Z}_n^*$. Wenn $a^{(n-1)/2} \equiv 1 \pmod{n}$ (das gilt wie gesagt für die Hälfte der Zahlen in $\mathbb{Z}_n^*$), so sollen die zwei Quadratwurzeln von a berechnet werden.

- Besonders einfach ist dies, wenn $n \equiv 3 \pmod{4}$.
- Dann ist $n + 1$ durch 4 teilbar.
- In diesem Fall nennt man n eine Blum-Primzahl.
- Man berechnet $a^{(n+1)/4}$.
- $a^{(n+1)/4}$ ist Quadratwurzel von a, denn nimmt man $a^{(n+1)/4}$ zum Quadrat, so ergibt sich:

$$a^{(n+1)/2} \equiv a^{(n-1)/2} \cdot a \equiv 1 \cdot a \equiv a \pmod{n}$$

- Wenn b eine Quadratwurzel von a ist, so ist $n - b$ die zweite.

Quadratwurzeln berechnen

- Die Berechnung ist einfach im Fall $n \equiv 3 \pmod 4$.
- Schwieriger ist der Fall, wenn $n \equiv 1 \pmod 4$.
- Tatsächlich gibt es einen $O(k^4)$-Algorithmus für diesen Fall.
- In der Praxis versucht man diesen Fall einfach zu vermeiden, indem man die Primzahl als Blum-Primzahl wählt.
- Bemerkung: Die Fälle $n \equiv 0 \pmod 4$ und $n \equiv 2 \pmod 4$ können nicht auftreten, weil dann $n - 1$ ungerade wäre (n ist aber ungerade Primzahl).

Quadratwurzeln berechnen

Im Kontext von Primzahltests interessieren uns insbesondere die Quadratwurzeln der Zahl 1.

- Modulo einer Primzahl $n > 2$ gibt es genau die zwei Quadratwurzeln 1 und -1 (also $n - 1$).
- Wenn, modulo n, $b^2 = 1$ für eine Zahl b gilt, so ist $0 = b^2 - 1 = (b - 1)(b + 1)$.
- Deshalb muss b kongruent 1 oder -1 sein.

Quadratwurzeln berechnen

- Das Quadratwurzelziehen ist nicht nur modulo einer Primzahl n effizient möglich.
- Dies ist auch der Fall, wenn die Faktorisierung von $n = p \cdot q$ bekannt ist.
- Man verwendet den Chinesischen Restsatz (CR).

Rechenoperationen mit dem Chinesischen Restsatz

Betrachten wir zunächst allgemein die Rechenoperationen mit dem Chinesischen Restsatz.

- Es gelte $(a_1, \ldots, a_k) \circ\!\!-\!\!\bullet\ x$ sowie $(b_1, \ldots, b_k) \circ\!\!-\!\!\bullet\ y$.
- Die Modulzahlen seien $n_1, \ldots, n_k$ sowie $n = \prod_{i=1}^{k} n_i$.
- Das Symbol $\diamond$ stehe für eine der Grundrechenarten.
- Es soll $x \diamond y \bmod n$ berechnet werden.
- Dieses Ergebnis erhält man auch durch Rücktransformation gemäß Chinesischem Restsatz:

$$(a_1 \diamond b_1, \ldots, a_k \diamond b_k) \circ\!\!-\!\!\bullet\ (x \diamond y) \bmod n$$

- Auf der linken Seite sind die Berechnungen jeweils $\bmod n_i$ zu verstehen.

Quadratwurzeln berechnen

- Sei $n = p \cdot q$ für zwei verschiedene Primzahlen p und q.
- Es sollen die Quadratwurzeln von x mod n berechnet werden.
- Dazu muss x natürlich ein quadratischer Rest mod n sein.
- Dies ist genau dann der Fall, wenn x quadratischer Rest mod p und ebenso mod q ist.
- Man kann, wie oben gezeigt, effizient eine Quadratwurzel w_1 mod p und ebenso eine Quadratwurzel w_2 mod q berechnen.
- Die vier Quadratwurzeln von x ergeben sich dann durch Rücktransformation von (w_1, w_2), $(w_1, -w_2)$, $(-w_1, w_2)$, $(-w_1, -w_2)$.

Das Rabin-Verfahren: Schlüsselgenerierung

- Wähle zufällig zwei sehr große Primzahlen $p \neq q$ mit $p \equiv q \equiv 3 \pmod 4$.
- Das Verfahren funktioniert auch ohne die Kongruenzbedingung, aber diese beschleunigt die Entschlüsselung.
- Berechne $n = p \cdot q$
- Der öffentliche Schlüssel ist n.
- Der geheime Schlüssel ist (p, q).
- Beispiel: $p = 11$, $q = 23$ und $n = 253$.
- Beachte: $11 \equiv 3 \pmod 4$ und $23 \equiv 3 \pmod 4$.

Das Rabin-Verfahren: Verschlüsseln

- Alice besorgt sich den öffentlichen Schlüssel n von Bob.
- Sie verschlüsselt ihre Nachricht m (wobei m eine natürliche Zahl mit $0 \leq m < n$ ist) zu

$$c = m^2 \mod n$$

  und sendet c an Bob.
- Beispiel: Die Nachricht $m = 158$ wird verschlüsselt zu

$$158^2 \mod 253 = 170$$

Das Rabin-Verfahren: Entschlüsseln

- Bob entschlüsselt den Geheimtext c durch Wurzelziehen, unter Benutzung des geheimen Schlüssels (p, q).
- Bob berechnet also

$$m_p = c^{(p+1)/4} \bmod p, \qquad m_q = c^{(q+1)/4} \bmod q$$

- m_p und $p - m_p$ sind die beiden Quadratwurzeln von c in $\mathbb{Z}_p^*$.
- m_q und $q - m_q$ sind die beiden Quadratwurzeln von c in $\mathbb{Z}_q^*$.
- Bob stellt die vier Kongruenzgleichungssysteme

$$\begin{aligned} x &\equiv a_1 \pmod{p} \\ x &\equiv a_2 \pmod{q} \end{aligned}$$

  mit $a_1 \in \{m_p, p - m_p\}$ und $a_2 \in \{m_q, q - m_q\}$ auf.
- Er löst die Gleichungssysteme mithilfe des Chinesischen Restsatzes und erhält so die vier Lösungen zur Gleichung $x^2 = c \bmod n$. Eine davon ist der Klartext.

Das Rabin-Verfahren: Entschlüsseln

Im Beispiel mit $p = 11$, $q = 23$ und $c = 170$ berechnet Bob

$$m_p = c^{(p+1)/4} \bmod p = c^3 \bmod p = 4$$

$$m_q = c^{(q+1)/4} \bmod q = c^6 \bmod q = 3$$

- Probe: $m_p = 4$ ist Quadratwurzeln von 170 in $\mathbb{Z}_{11}^*$, denn $4^2 \equiv 5 \equiv 170 \pmod{11}$.
- Probe: $m_q = 3$ ist Quadratwurzeln von 170 in $\mathbb{Z}_{23}^*$, denn $3^2 \equiv 9 \equiv 170 \pmod{23}$.

Erinnerung: Chinesischer Restsatz für $n = p \cdot q$

Gesucht ist eine Lösung x für das Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv a_1 \pmod{p} \\ x &\equiv a_2 \pmod{q} \end{aligned}$$

Die Lösung ist $x = a_1 \cdot y_1 \cdot q + a_2 \cdot y_2 \cdot p$, wobei

- y_1 ist invers zu q modulo p
- y_2 ist invers zu p modulo q.

Mit dem erweiterten Euklid-Algorithmus berechnet Bob die multiplikativ Inversen für den CR-Algorithmus:

- $y_1 = 1$ ist invers zu $q = 23$ modulo $p = 11$
- $y_2 = 21$ ist invers zu $p = 11$ modulo $q = 23$.

Das Rabin-Verfahren: Entschlüsseln

Für das Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv 4 \pmod{p} \\ x &\equiv 3 \pmod{q} \end{aligned}$$

gibt der CR-Algorithmus $4 \cdot 1 \cdot 23 + 3 \cdot 21 \cdot 11 = 785 \mod 253$, also 26 aus.

Für das Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv 4 \pmod{p} \\ x &\equiv 20 \pmod{q} \end{aligned}$$

gibt der CR-Algorithmus $4 \cdot 1 \cdot 23 + 20 \cdot 21 \cdot 11 = 4712 \mod 253$, also 158 aus.

Die vier Quadratwurzeln von 170 mod 253 in $\{1, \ldots, 252\}$ sind 26, $253 - 26 = 227$, 158 und $253 - 158 = 95$.
Eine dieser Quadratwurzeln ist der Klartext.

Das Rabin-Verfahren

Die wichtigste Methode, um aus den vier Quadratwurzeln den richtigen Klartext auszusuchen, ist folgende:

- Man fügt dem Klartext eine zuvor verabredete Redundanz hinzu.
- Man kann z.B. die ersten oder letzten 64 Bits der Nachricht wiederholen.
- Dann wird mit hoher Wahrscheinlichkeit nur einer der vier möglichen Klartexte diese Redundanz besitzen.
- Bob wird genau diesen als den gesendeten Klartext identifizieren.
- Wenn keine der Quadratwurzeln diese Eigenschaft hat, wird er c als Fälschung ablehnen.

Ausblick zur Rabin-Kryptoanalyse

- Es sei n das Produkt zweier Primzahlen p und q.
- Es sei $a \in \mathbb{Z}_n^*$ ein (beliebiger) quadratischer Rest modulo n.
- Das Problem, bei unbekannten Zahlen p und q eine Quadratwurzel modulo n von a zu bestimmen, ist äquivalent zu dem Problem, die Primfaktoren p und q von n zu berechnen (der Beweis folgt später).
- Die Schwierigkeit, das Rabin-Verfahren zu brechen, beruht also wiederum auf der Schwierigkeit des Faktorisierungsproblems.

Primzahlen

Primzahlsatz: Für jede natürliche Zahl n gibt es ungefähr $n/\ln(n)$ viele Primzahlen $\leq n$.

- Dies bedeutet, dass eine zufällig aus der Menge $\{1, 2, \ldots, n\}$ gezogene Zahl mit Wahrscheinlichkeit $1/\ln(n)$ eine Primzahl ist.
- Die Wahrscheinlichkeit erhöht sich auf $2/\ln(n)$, wenn man ausschließlich ungerade Zahlen zieht.
- Zieht man beispielsweise eine ungerade Zahl aus $\{1, 3, \ldots, 2^{256} - 1\}$ zufällig, dann ist sie mit Wahrscheinlichkeit $2/\ln(2^{256}) \approx 1/88.7$ eine Primzahl.
- Mit anderen Worten: Im Schnitt muss man 89 ungerade Zahlen unter Gleichverteilung ziehen, um eine Primzahl aus obiger Menge zu finden.

Erster probabilistischer Primzahltest (Fermat-Test)

Satz von Fermat: Wenn n eine Primzahl ist, so gilt für alle $a \in \mathbb{Z}_n^*$

$$a^{n-1} \equiv 1 \pmod{n}$$

Korollar: Existiert ein $a \in \mathbb{Z}_n^*$ mit $a^{n-1} \not\equiv 1 \pmod{n}$, dann ist n mit Sicherheit keine Primzahl.

```
PROC FermatTest( n )
a := random( 1, n − 1 )
IF ggT(a, n) ≠ 1 THEN RETURN "keine Primzahl"
IF a^(n−1) ≢ 1 (mod n) THEN RETURN "keine Primzahl"
RETURN "(vermutlich) Primzahl"
```

Carmichael-Zahlen

Definition: Eine zusammengesetzte Zahl n nennt man Carmichael Zahl, falls

$$a^{n-1} \equiv 1 \pmod{n}$$

für alle $a \in \mathbb{Z}_n^*$ gilt.

- Carmichael-Zahlen bestehen den Fermat-Test fast immer (Ausnahme: $\mathrm{ggT}(a, n) \neq 1$).
- Es gibt unendlich viele Carmichael Zahlen.
- Die Folge der Carmichael Zahlen beginnt mit

$$561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, \ldots$$

- Carmichael Zahlen sind extrem selten.
- Es gibt nur 255 Carmichael Zahlen, die kleiner als 10^8 sind.

Quadratwurzeln der 1

- Das Ziel ist es, den Fermat-Test so zu verbessern, dass auch Carmichael Zahlen als zusammengesetzt erkannt werden.
- Dieses Ziel wird durch den "Quadratwurzeln der 1"-Test erreicht.
- Erinnerung: Falls $n > 2$ eine Primzahl ist, dann sind 1 und $n - 1$ die einzigen Quadratwurzeln von 1 (mod n) in $\mathbb{Z}_n^*$.
- Folgerung: Besitzt n eine Quadratwurzel x von 1 (mod n) mit $x \not\equiv \pm 1 \pmod{n}$, dann ist n keine Primzahl.

Probabilistischer Primzahltest (nach Miller-Rabin)

Die auf Primzahleigenschaft zu testende Zahl n sei ungerade, hat also die Form $n = 1 + 2^r \cdot u$, u ungerade, $r \geq 1$.

```
PROC MillerRabinTest( n )
a := random( 1, n − 1 )
IF ggT(a, n) ≠ 1 THEN RETURN "keine Primzahl"
d := a^u mod n = modexp( a, u, n )
IF d = 1 THEN RETURN "(vermutlich) Primzahl"
FOR i := 0 TO r − 1 DO
    IF d = −1 THEN RETURN "(vermutlich) Primzahl"
    ELSE d := d^2 mod n
RETURN "keine Primzahl"
```

Zahlenbeispiel

- Wir wenden den Miller-Rabin-Primzahltest auf die Carmichaelzahl $561 = 3 \cdot 11 \cdot 17$ an.
- Wir wählen $a = 2$.
- Es gilt $\mathrm{ggT}(2, 561) = 1$ und $560 = 2^4 \cdot 35$.
- Der Algorithmus "testet" die Zahlen $a^u, a^{2u}, \ldots a^{2^{r-1}u}$, also

$$2^{35} \equiv 263, 2^{70} \equiv 166, 2^{140} \equiv 67, 2^{280} \equiv 1 \pmod{n}$$

- Der Algorithmus gibt "keine Primzahl" aus.
- Man sieht, dass 67 eine Quadratwurzel der 1 modulo 561 ist. Das beweist, dass 561 keine Primzahl ist.

Korrektheit des Miller-Rabin-Primzahltests

Satz: Wenn der Miller-Rabin-Primzahltest "keine Primzahl" ausgibt, so ist die Eingabezahl n tatsächlich keine Primzahl.

Beweis: Falls $\mathrm{ggT}(a, n) \neq 1$, so ist n sicherlich keine Primzahl. Wir nehmen nun an, dass der Algorithmus "keine Primzahl" ausgibt, obwohl die Eingabezahl n eine Primzahl ist (und erhalten einen Widerspruch).

- Da die Ausgabe "keine Primzahl" lautet, muss $a^u \not\equiv 1 \bmod n$ gelten.
- Nun betrachten wir die Folge der Zahlen $a^u, a^{2u}, \ldots a^{2^{r-1}u}$, die d annimmt (diese werden in der Schleife getestet).
- Da der Algorithmus "keine Primzahl" ausgibt, muss für alle $i \in \{0, \ldots, r-1\}$ gelten: $a^{2^i u} \not\equiv -1 \pmod{n}$.

Korrektheit des Miller-Rabin-Primzahltests

- Da wir angenommen hatten, dass n eine Primzahl ist, folgt

$$a^{2^r u} = a^{n-1} \equiv 1 \pmod{n}$$

  mit dem Satz von Fermat.
- D.h. $a^{2^{r-1} u}$ ist eine Quadratwurzeln von 1 modulo n, also

$$a^{2^{r-1} u} \equiv \pm 1 \pmod{n}$$

- Es muss $a^{2^{r-1} u} \equiv 1 \pmod{n}$ gelten, weil der Fall $a^{2^{r-1} u} \equiv -1 \pmod{n}$ ausgeschlossen wurde.
- Also ist $a^{2^{r-2} u}$ ist eine Quadratwurzeln von 1 modulo n.
- Mit demselben Argument folgt $a^{2^{r-1} u} \equiv 1 \pmod{n}$.
- Durch wiederholtes Anwenden des Argumentes folgt schließlich $a^u \equiv 1 \pmod{n}$.
- Dieser Widerspruch beweist den Satz.

Probabilistischer Primzahltest (nach Miller-Rabin)

- Bei Ausgabe "keine Primzahl" ist n also sicherlich keine Primzahl.
- Man kann zeigen, dass die Fehlerwahrscheinlichkeit (der Algorithmus gibt "(vermutlich) Primzahl" aus, obwohl die Eingabezahl n keine Primzahl ist) höchstens $1/4$ beträgt (Buchmann, Theorem 8.4.3).
- Bei Ausgabe "(vermutlich) Primzahl" ist n also mit Wahrscheinlichkeit $\geq 3/4$ tatsächlich eine Primzahl.
- Man kann die Fehlerwahrscheinlichkeit von $1/4$ bei t-facher Wiederholung des Tests unter $(1/4)^t$ drücken.
- Die Laufzeit des Primzahltests, bei einer Eingabe n mit k Bits, ist $O(k^3)$.

Probabilistischer Primzahltest (nach Miller-Rabin)

- Im Jahr 2002 wurde von Agrawal, Kayal, Saxena ein deterministischer, polynomialer Primzahltest publiziert (AKS-Algorithmus).
- Der Grund, dass man immer noch den probabilistischen Miller-Rabin-Primzahltest bevorzugt ist, dass die Laufzeit von AKS in der Größenordnung von $O(k^{12})$ liegt.

Zufallsprimzahl in Binärdarstellung herstellen herstellen

Es soll eine k-Bit Zufallsprimzahl hergestellt werden.

- Man setzt das erste und das letzte Bit auf 1 (damit die Zahl ungerade wird).
- Man wählt die restlichen $k-2$ Bits dazwischen zufällig und führt einen Primzahltest durch.
- Sofern diese Zahl keine Primzahl ist, wiederholt man das Verfahren.
- Der Primzahlsatz besagt, dass es etwa $n/\ln(n)$ viele Primzahlen $\leq n$ gibt.
- Konkret: Um eine 500-Bit Zufallsprimzahl herzustellen, benötigt man etwa 173 Versuche; um eine 1000-Bit Zufallsprimzahl herzustellen, benötigt man etwa 347 Versuche; allgemein etwa $0.35 \cdot k$ Versuche.

Zufallsprimzahl in Binärdarstellung herstellen herstellen

Es soll eine k-Bit Zufallsprimzahl hergestellt werden.

- Im Sinne der O-Notation betrachtet hat ein Primzahltest die Laufzeit $O(k^3)$, da dieser ähnlich aufgebaut ist wie eine modulare Exponentiation (und da die notwendige Anzahl von Wiederholungen als konstant betrachtet werden kann).
- Ferner benötigt man aufgrund des Primzahlsatzes im Erwartungswert größenordnungsmäßig $\ln n = O(k)$ Versuche, bis man eine Primzahl findet.
- Das macht zusammen die Laufzeit $O(k^4)$, um eine Zufallsprimzahl mit k Bits herzustellen.

Anmerkungen zu probabilistischen Algorithmen

- Wir haben bereits einige probabilistischen Algorithmen kennengelernt.
- Probabilistische Algorithmen machen den Rechenablauf, zumindest an manchen Stellen, abhängig von einer Zufallszahl.
- Insofern können, bei derselben Eingabe, verschiedene Rechenabläufe, und auch Rechenergebnisse, entstehen.
- Man kann probabilistische Algorithmen wie folgt klassifizieren:
  - Monte Carlo-Algorithmen
  - Las Vegas-Algorithmen

Monte Carlo-Algorithmen

- Wir betrachten Algorithmen, die eines von zwei möglichen Ergebnissen liefern sollen (Entscheidungsverfahren).
- Ein Monte Carlo-Algorithmus kann mit einer gewissen Wahrscheinlichkeit p ein falsches Ergebnis berechnen.
- Beim Miller-Rabin-Test ist das z.B. der Fall.
  - Im Fall einer Primzahl als Eingabe wird immer das Ergebnis "Primzahl" erscheinen.
  - Im Fall einer Nicht-Primzahl kann mit einer Wahrscheinlichkeit $p \leq 1/4$ doch "Primzahl" festgestellt werden.
  - Das heißt, die Fehlermöglichkeit ist nur einseitig gegeben.
- Man wiederholt den Algorithmus bei derselben Eingabe t-mal.
- Sollte nur einmal "keine Primzahl" konstatiert werden, so ist die Eingabezahl mit Sicherheit keine Primzahl.
- Die Wahrscheinlichkeit, dass das die Ausgabe "Primzahl" t-mal falsch war, beträgt dann nur noch p^t.

Anmerkungen zu probabilistischen Algorithmen

- Bei einem sinnvollen probabilistischen Algorithmus kommt es darauf an, dass die Wahrscheinlichkeit, ein korrektes Ergebnis zu liefern nicht exponentiell klein sein darf (sonst wären exponentiell viele Wiederholungen nötig).
- Sie muss wie z.B. wie bei Miller-Rabin konstant (nämlich $3/4$), oder zumindest $1/p(n)$ sein, wobei p ein Polynom ist.
- Bei solchen sinnvollen probabilistischen Algorithmen kann die Wahrscheinlichkeit, kein Ergebnis oder ein falsches Ergebnis zu erhalten, durch entsprechende Wiederholung exponentiell klein gehalten werden.
- Sofern die Laufzeit eines probabilistischen Algorithmus einschließlich der notwendigen Wiederholungszahl polynomial ist, kann man diesen Algorithmus als gleichwertig zu einem deterministischen polynomialen Algorithmus betrachten.

Las Vegas-Algorithmen

- Ein Las Vegas-Algorithmus darf kein falsches Ergebnis berechnen; er darf lediglich Misserfolg konstatieren.
- Die Wahrscheinlichkeit des Misserfolgsfalls sei p.
- Dann kann der Algorithmus mehrfach, bei derselben Eingabe, mit unabhängigen Zufallszahlen, wiederholt werden.
- Die mittlere Anzahl Wiederholungen, bis eine Lösung gefunden wird, beträgt dann gemäß geometrischer Verteilung $1/(1-p)$.
- Ein Beispiel ist der Algorithmus, der bei einer sicheren Primzahl eine Primitivwurzel finden soll.
- Mit Wahrscheinlichkeit $1/2$ kann Misserfolg auftreten.
- Im Erwartungswert muss man es also 2-mal versuchen, bis man eine Primitivwurzel gefunden hat.

Erinnerung: Primitivwurzel bei sicherer Primzahl bestimmen

- Eine Primzahl n ist eine sichere Primzahl, wenn $n = 2q + 1$ für eine Primzahl q. Beachte: $\mathbb{Z}_n^*$ ist zyklisch.
- Die Faktorisierung $n - 1 = 2 \cdot q$ steht für das folgende Primitivwurzel-Kriterium zur Verfügung.
- Es gilt: $a \in \mathbb{Z}_n^*$ ist genau dann eine Primitivwurzel modulo n, wenn $a^{(n-1)/p} \neq 1$ für alle Primzahlen p, die $n - 1$ teilen.
- Die Anzahl der Primitivwurzeln der zyklischen Gruppe $\mathbb{Z}_n^*$ beträgt $\varphi(\varphi(n)) = \varphi(n - 1) = \varphi(2q) = q - 1$.
- Man wählt a mehrfach zufällig, und wendet das Primitivwurzel-Kriterium an.
- Da $q - 1 = (n - 3)/2$ findet man eine Primitivwurzel nach wenigen (nämlich ca. 2) Zufallsversuchen.

Die Klasse P

- Die Klasse P umfasst alle algorithmischen Aufgabenstellungen, für die es einen (deterministischen) Algorithmus gibt, dessen Rechenzeit als Funktion der Eingabelänge n *polynomial* ist.
- Man schreibt $f(n) = O(g(n))$, wenn Folgendes gilt:

$$\exists\, n_0 \in \mathbb{N}\ \exists\, c > 0\ \forall\, n \geq n_0\ : f(n) \leq c \cdot g(n)$$

- Bei Zahlen als Eingabe, wie es in diesem Kontext fast ausschließlich der Fall ist, ist als Eingabelänge die Anzahl der Bits zu verstehen, die die Zahl in Binärdarstellung ausmacht.
- Man muss beachten, dass ein exponentieller (umgekehrt ein logarithmischer) Zusammenhang besteht:

$$\text{Zahlenwert} \approx 2^{\text{Bitlänge}}, \qquad \text{Bitlänge} \approx \log_2(\text{Zahlenwert})$$

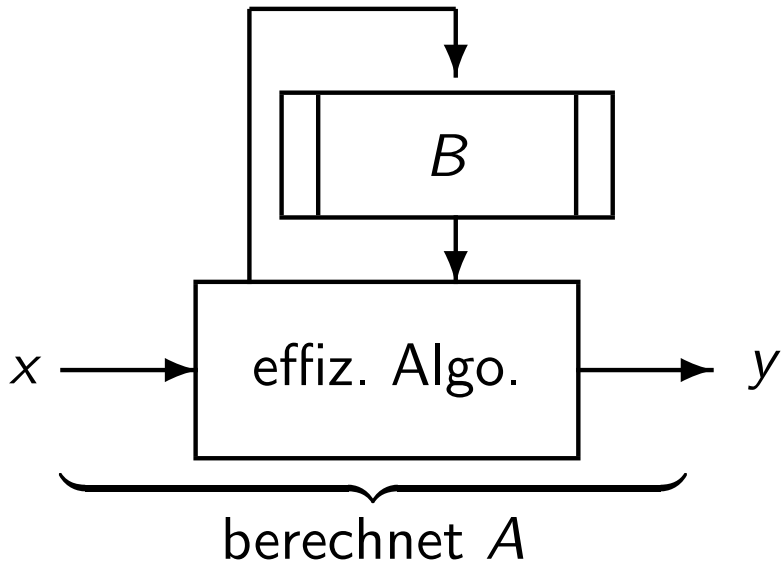
Die Klasse P

- Die Eigenschaft "polynomiale Laufzeit" wird vereinfachend mit "ist effizient berechenbar" gleichgesetzt.
- Im Einzelfall kann es aber durchaus eine Rolle spielen, ob man einen $O(n)$ oder einen $O(n^3)$ Algorithmus zur Verfügung hat, obwohl beide polynomial sind.
- Die Sicherheit kryptographischer Verfahren hängt weitgehend davon ab, dass es bestimmte algorithmische Aufgabenstellungen gibt (Musterbeispiel: das Faktorisieren einer Zahl), die nur mit exponentiellem Rechenaufwand zu bewältigen sind, also Rechenaufwand c^n für eine Konstante $c > 1$ benötigen.
- Leider gibt es keinen Beweis, dass dies so ist, sondern nur die Beobachtung, dass es für solche Probleme noch nicht gelungen ist, effiziente Algorithmen zu finden.

Orakel-Reduzierbarkeit

- Ein algorithmisches Problem A ist auf anderes Problem B *effizient (Orakel-) reduzierbar*, symbolisch $A \preceq B$, falls es einen effizienten (d.h. polynomialen) Basisalgorithmus gibt, der das Problem A löst bzw. berechnet.
- Dieser Algorithmus darf ein Unterprogramm verwenden und ggf. mehrfach mit verschiedenen Eingaben aufrufen, welches das Problem B löst.
- Die Komplexität, also notwendige Rechenzeit zur Berechnung von B bleibt unspezifiziert und fließt nicht in die Laufzeit des Basisalgorithmus ein.
- Im Skript wird nicht unterschieden zwischen deterministisch effizient reduzierbar und probabilistisch effizient reduzierbar (mit beliebig kleinem Fehler).

Orakel-Reduzierbarkeit



Orakel-Reduzierbarkeit

- Die Definition ist auch auf Funktionen anwendbar anstelle von Entscheidungsproblemen (sowohl was A betrifft als auch was B betrift).
- Die Relation $\preceq$ ist reflexiv (denn es gilt $A \preceq A$), und transitiv (falls $A \preceq B$ gilt sowie $B \preceq C$, so folgt durch Zusammenfügen der betreffenden Basisalgorithmen, dass $A \preceq C$).

Orakel-Reduzierbarkeit

Satz: Sofern $A \preceq B$ gilt und $B \in \mathrm{P}$, so folgt $A \in \mathrm{P}$, bzw. äquivalent dazu: sofern $A \preceq B$ und $A \notin \mathrm{P}$, so folgt $B \notin \mathrm{P}$.

Beweis:

- Bei Eingabelänge $|x| = k$ habe der Basisalgorithmus die polynomiale Laufzeit $p(k)$.
- Dann kann auch die Anzahl der Orakel-Aufrufe sowie die Länge der Orakelfragen höchstens $p(k)$ sein.
- Da das Orakel in P liegt, gibt es für das Orakel einen polynomialen Algorithmus mit Laufzeitpolynom q.
- Insgesamt kann die Laufzeit beider Algorithmen zusammen höchstens $p(k) + p(k) \cdot q(p(k))$, also polynomial, sein.

NP und NP-vollständig

- Die Klasse NP ist nur für Entscheidungsprobleme (ja/nein-Probleme) definiert.
- Ein Entscheidungsproblem A (also zu entscheiden, ob die Eingabe x Element von A ist oder nicht) liegt in NP, wenn es einen polynomialen Algorithmus M (wie bei der Definition von P) gibt, allerdings mit zwei Eingaben: x (die eigentliche Eingabe) und y, so dass Folgendes für alle Eingaben x gilt:

$$x \in A \Rightarrow \exists\, y : M(x, y) = 1$$

$$x \notin A \Rightarrow \forall\, y : M(x, y) = 0$$

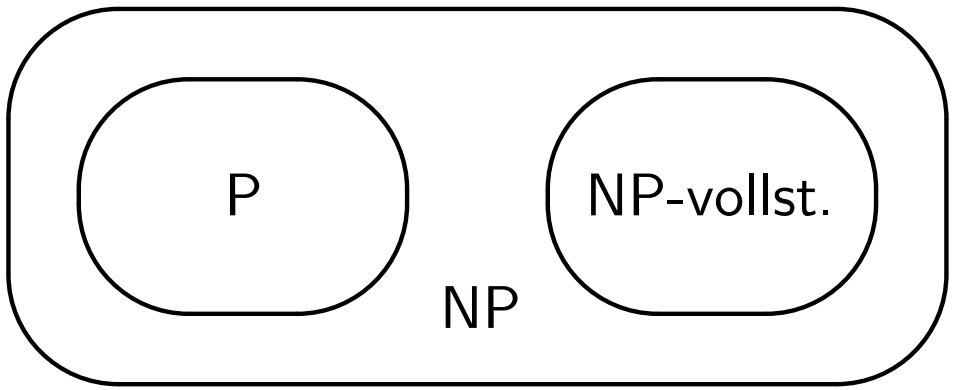
- Eine zu beachtende Nebenbedingung ist, dass die Zusatzeingabe y höchstens polynomial länger ist als x, kurz: $|y| \leq poly(|x|)$.

NP und NP-vollständig

- Etwas informal kann man sagen, NP umfasst diejenigen Problemstellungen, bei denen man eine Lösung (bzw. Lösungsvorschlag), das ist y bei der obigen Definition, effizient überprüfen kann.
- Zum Beispiel liegt das Entscheidungsproblem, ob $x \in \mathbb{N}$ eine zusammengesetzte Zahl ist, in NP, denn wenn zusätzlich ein y vorgelegt wird, so kann man effizient überprüfen, ob y ein nicht-trivialer Teiler von x ist.

NP und NP-vollständig

- Die Klasse NP umfasst die Klasse P, enthält jedoch auch viele Probleme, für die bisher keine polynomialen Algorithmen bekannt sind.
- Insbesondere betrifft dies die Menge der NP-vollständigen Probleme.
- Die Skizze zeigt die vermutete Situation $P \neq NP$.



NP und NP-vollständig

- Ein Entscheidungsproblem L in NP ist NP-vollständig, wenn für jedes Problem A in NP die Relation $A \preceq L$ besteht.
- Inhaltlich bedeutet dies, dass L mindestens so schwierig zu lösen ist wie A.
- Es handelt sich also um einen relativen Vergleich zwischen zwei algorithmischen Problemstellungen (ohne damit eine absolute Komplexitätsaussage zu machen, also polynomial oder exponentiell).
- Insbesondere heißt das, dass jedes einzelne NP-vollständige Problem L bereits das P-NP-Problem lösen kann: Es gilt $\mathrm{P} = \mathrm{NP}$ genau dann, wenn $L \in \mathrm{P}$.

Einwegfunktionen

- Eine *Einwegfunktion* ist eine Funktion, die in Vorwärtsrichtung effizient berechenbar ist, nicht jedoch in Rückwärtsrichtung:

$$f \in \mathrm{P} \quad \text{und} \quad f^{-1} \notin \mathrm{P}$$

- In manchen Anwendungen muss f injektiv sein (z.B. beim Verschlüsseln von Klartexten), dann ist für jedes $y = f(x)$ das Urbild x eindeutig: $x = f^{-1}(y)$.
- Wir wenden das Konzept der Einwegfunktion aber auch auf nicht-injektive Funktionen (etwa Hashfunktionen) an.
- Dann ist mit $f^{-1} \notin \mathrm{P}$ gemeint, dass es mit polynomialen Aufwand unmöglich ist, irgendein Urbild x von y zu finden.

Einwegfunktionen

- Eine *Einwegfunktion* ist eine Funktion, die in Vorwärtsrichtung effizient berechenbar ist, nicht jedoch in Rückwärtsrichtung:

$$f \in \mathrm{P} \quad \text{und} \quad f^{-1} \notin \mathrm{P}$$

- Ferner muss die Länge von x mit der von y in einem polynomialen Zusammenhang stehen: $|x| \leq \mathit{poly}(|y|)$ und $|y| \leq \mathit{poly}(|x|)$. Dann gilt (ohne Beweis):

$$\text{Es gibt Einwegfunktionen} \quad \Rightarrow \quad \mathrm{P} \neq \mathrm{NP}$$

  D.h. wir wissen zurzeit nicht, ob Einwegfunktionen überhaupt existieren.

- Im Folgenden werden wir auch Funktionen, zu denen bisher keine in angemessener Zeit praktisch ausführbare Umkehrung bekannt ist, als Einwegfunktion bezeichnen.

Einwegfunktionen mit Falltür

- Eine *Einwegfunktion mit Falltür* (trapdoor one-way function) ist eine Einwegfunktion f, so dass es einen effizienten Alg. M und für jede Länge n (von y) einen Schlüssel k_n gibt, so dass für alle y der Länge n gilt: $M(y, k_n) = f^{-1}(y)$.
- Das heißt, für den Berechtigten, der im Besitz des Schlüssels k_n ist, ist es effizient möglich, die Umkehrfunktion von f zu berechnen.
- Ein Beispiel hierzu findet sich beim RSA-Kryptosystem.
- Öffentliche Informationen sind hier der Modul n und der Verschlüsselungsexponent e.
- Geheim sind die Faktorisierung von $n = p \cdot q$ und der Entschlüsselungsexponent d.
- Die Einwegfunktion ist hier $e \mapsto d$. Mit Hilfe des Schlüssels p, q lässt sich mittels extggT$((p-1)(q-1), e)$ das Inverse d zu e in $\mathbb{Z}^*_{\varphi(n)} = \mathbb{Z}^*_{(p-1)(q-1)}$ effizient bestimmen.

RSA-Sicherheit

Wenn der RSA-Modul n, der Verschlüsselungsexponent e und die Chiffre $c = m^e \mod n$ bekannt sind, so kann man verschiedene Probleme formulieren:

- B_1 Bestimme den Klartext m.
- B_2 Berechne den Entschlüsselungsexponenten d.
- B_3 Berechne $\varphi(n)$.
- B_4 Bestimme einen Faktor p (und damit auch q) von n.

RSA-Sicherheit

Es ist klar, dass folgende Orakel-Reduzierbarkeiten gelten:

$$B_1 \preceq B_2 \preceq B_3 \preceq B_4$$

- Wenn das Orakel die Faktoren p und q von n liefert, so kann man $\varphi(n) = (p-1)(q-1)$ in polynomialer Zeit berechnen.
- Wenn das Orakel $\varphi(n)$ liefert, so kann man den Entschlüsselungsexponenten d mithilfe des erweiterten Euklid-Algorithmus in polynomialer Zeit berechnen.
- Wenn das Orakel den Entschlüsselungsexponenten d liefert, so kann man mithilfe der modularen Exponentiation den Klartext m in polynomialer Zeit berechnen.

RSA-Sicherheit: $B_4 \preceq B_3$

Lemma: Es gilt $B_4 \preceq B_3$.

Beweis:

- Wir wissen, dass n die Form pq hat.
- Das Orakel liefert die Zahl $k = \varphi(n)$ und wir wissen, dass k die Form $(p-1)(q-1)$ hat.
- $k = (p-1)(q-1) = pq - p - q + 1 = n - p - n/p + 1$.
- Multiplikation mit p ergibt: $kp = np - p^2 - n + p$.
- Somit erhalten wir die quadratische Gleichung $p^2 + (k - n - 1) \cdot p + n = 0$.
- Nach p auflösen ergibt die beiden Primfaktoren.

Beispiel: $B_4 \preceq B_3$

Beispiel 1:

- Sei $n = 323$.
- Das Orakel liefert die Zahl $k = \varphi(n) = 288$.
- Wir erhalten die quadratische Gleichung $p^2 - 36 \cdot p + 323 = 0$.
- Nach p auflösen ergibt die beiden Primfaktoren

$$18 \pm \sqrt{18^2 - 323} = 18 \pm 1$$

d.h. $n = 17 \cdot 19$.

Beispiel 2:

- Sei $n = 84773093$.
- Das Orakel liefert die Zahl $k = \varphi(n) = 84754668$.
- Wir erhalten die quadratische Gleichung $p^2 - 18426 \cdot p + 84773093 = 0$.
- Nach p auflösen ergibt die Primfaktoren 9539 und 8887.

RSA-Sicherheit: $B_4 \preceq B_2$

Während das schwache Brechen von RSA, nämlich $(n, e, c) \mapsto m$ berechnen, zwar auf das Faktorisieren von n reduzierbar ist (also $B_1 \preceq B_4$), ist die Umkehrrichtung nicht bewiesen.

Die Umkehrrichtung gilt jedoch für das starke Brechen von RSA, nämlich das Berechnen von $(n, e) \mapsto d$ (also $B_4 \preceq B_2$).

Das wollen wir im Folgenden beweisen.

Wir zeigen also: Faktorisieren $\preceq$ RSA im starken Sinne brechen.

Erinnerung: Quadratwurzeln der Zahl 1

- Modulo einer Primzahl $n > 2$ gibt es genau die zwei Quadratwurzeln 1 und -1 (also $n-1$).
- Wenn, modulo n, $b^2 = 1$ für eine Zahl b gilt, so ist $0 = b^2 - 1 = (b-1)(b+1)$.
- Da in dem Körper $(\mathbb{Z}_n, +, *)$ das Polynom $x^2 - 1$ höchstens zwei Nullstellen hat, muss b kongruent 1 oder -1 sein.

Erinnerung: Quadratwurzeln berechnen

Satz: Sei n das Produkt zweier Primzahlen p und q. Dann besitzt $x^2 \equiv 1 \pmod{n}$ genau vier Lösungen.

Beweis: Sei x eine Lösung der Gleichung $x^2 \equiv 1 \pmod{n}$. Dann folgt, dass $x \bmod p$ eine Lösung von (a) und $x \bmod q$ eine Lösung von (b) ist:

$$\begin{array}{lrcl} (a) & x^2 & \equiv & 1 \mod p \\ (b) & x^2 & \equiv & 1 \mod q \end{array}$$

- Die Gleichung (a) besitzt die zwei Lösungen $1, p-1 \in \mathbb{Z}_p^*$.
- Die Gleichung (b) besitzt die zwei Lösungen $1, q-1 \in \mathbb{Z}_q^*$.

Erinnerung: Quadratwurzeln berechnen

Satz: Sei n das Produkt zweier Primzahlen p und q. Dann besitzt $x^2 \equiv 1 \pmod{n}$ genau vier Lösungen.

Beweis (Fortsetzung): Gemeinsame Lösungen ergeben sich nach dem Chinesischen Restsatz durch

$$\begin{aligned} x &\equiv a_1 \bmod p \\ x &\equiv a_2 \bmod q \end{aligned}$$

wobei $a_1 \in \{1, p-1\}$ und $a_2 \in \{1, q-1\}$.

- Die Kombination von jeweils zwei Lösungen ergibt vier Gleichungssysteme, die genau eine Lösung in $\mathbb{Z}_n^*$ besitzen.
- Die Lösungen sind verschieden, da die rechten Seiten der vier Gleichungssysteme verschieden sind.

Erinnerung: Quadratwurzeln berechnen

- Für $n = p \cdot q$, ergeben sich die vier Quadratwurzeln der 1 also durch Rücktransformation von $(1, 1)$, $(1, -1)$, $(-1, 1)$ und $(-1, -1)$.
- Zwei der rücktransformierten Werte sind 1 und -1 $(= n - 1)$.
- Die anderen beiden Werte a und $-a$ sind von 1 und -1 verschieden.

Faktorisieren $\preceq$ RSA im starken Sinne brechen

- Das Orakel liefert zu e dasjenige d mit $ed \equiv 1 \bmod \varphi(n)$.
- Es gilt also $ed - 1 = k\varphi(n)$ für eine ganze Zahl k.
- Wir kennen $\varphi(n)$ nicht, wissen aber, dass es eine gerade Zahl ist (sogar durch 4 teilbar), weil $p-1$ und $q-1$ gerade Zahlen sind und $\varphi(n) = (p-1)(q-1)$.
- Also ist auch $g := ed - 1$ eine gerade Zahl.

Unter dieser Voraussetzung können wir die Faktorisierung von n mithilfe des folgenden probabilistischen Algorithmus berechnen.

Probabilistischer Algorithmus

- Wähle (ggf. mehrfach) x zufällig mit $1 < x < n$.
- Berechne $\mathrm{ggT}(x, n)$. Falls das Ergebnis ungleich 1 ist, so gib dieses als nicht-trivialen Faktor zurück und terminiere.
- Sonst gilt $x \in \mathbb{Z}_n^*$ und $x^{\varphi(n)} \equiv 1 \pmod{n}$ (Satz von Euler).
- Es folgt $x^g \equiv 1 \pmod{n}$, weil $g = ed - 1 = k\varphi(n)$.
- Da g eine gerade Zahl ist, kann man $x^{g/2}$ mod n bilden.
- Das Ergebnis y ist dann eine Quadratwurzel der 1 mod n.
- Es gibt vier Möglichkeiten: $y = \pm 1$ oder $y = \pm a$ mit $a \neq 1$.
- Falls $y = \pm a$, so gib $\mathrm{ggT}(y - 1, n)$ zurück und terminiere.
- Wenn $y = -1$, so wähle ein neues x.
- Sofern $y = 1$ und der Exponent immer noch eine gerade Zahl ist, kann man den Prozess iterieren, also den Exponenten halbieren, und damit eine Quadratwurzel der 1 erhalten.

Beispiel: Probabilistischer Algorithmus

- Der öffentliche Schlüssel sei $(n, e) = (253, 3)$.
- Das Orakel liefert $d = 147$ mit $ed = 3 \cdot 147 \equiv 1 \mod \varphi(n)$.
- $g = ed - 1 = 441 - 1 = 440$.
- x wird zufällig gewählt, z.B. $x = 2$.
- Es gilt: ggT(x, n) =ggT$(2, 253) = 1$.
- Damit folgt: $2^{440} \mod 253 = 1$
- Der Algorithmus berechnet:
  - $2^{220} \mod 253 = 1$
  - $2^{110} \mod 253 = 1$
  - $2^{55} \mod 253 = 208$
- Der Algorithmus gibt ggT$(208 - 1, 253) = 23$ zurück.
- Der andere Primfaktor ist ggT$(208 + 1, 253) = 11$.
- Die Faktorisierung lautet $253 = 11 \cdot 23$.

Analyse des Algorithmus

- Wähle $r, u \in \mathbb{N}$, so dass $r \geq 1$, u ungerade und $g = 2^r \cdot u$.
- Der Algorithmus betrachtet die Folge $x^{2^{r-1} \cdot u}, \ldots, x^{2u}, x^u$.
- Wenn die Folge aus lauter Einsen besteht, so wird ein neues x gewählt.
- Anderenfalls sei y das erste Element der Folge mit $y \neq 1$.
- Wenn $y = -1$, so wird ein neues x gewählt.
- Sonst gilt $y = \pm a$ mit $a \neq 1$.
- Man kann zeigen, dass $y = \pm a$ mit Wahrscheinlichkeit wenigstens $1/2$ auftritt (Buchmann, S.141-142).
- D.h. die Erfolgswahrscheinlichkeit des Algorithmus ist in jeder Iteration wenigstens $1/2$.

Korrektheit des Algorithmus

- Im Erfolgsfall gilt $y = \pm a$ mit $a \neq 1$ und $y^2 \equiv 1 \pmod{n}$.
- D.h. $y^2 - 1 = (y-1)(y+1) = mn$ für eine ganze Zahl m.
- Da n das Produkt $(y-1)(y+1)$ teilt, muss dies auch für p und q gelten.
- Da n weder $(y-1)$ noch $(y+1)$ teilt, muss p genau einen der Faktoren teilen und q den anderen.
- Mittels $\mathrm{ggT}(y-1, n)$ bzw. $\mathrm{ggT}(y+1, n)$ ergibt sich also die Faktorisierung von n.

Shor-Algorithmus zum Faktorisieren

Das Faktorisierungsproblem lässt sich auf das Problem, die Ordnung von Elementen zu bestimmen, reduzieren.

- Wähle (ggf. mehrfach) x zufällig mit $1 < x < n$.
- Berechne $\mathrm{ggT}(x, n)$. Falls das Ergebnis ungleich 1 ist, so gib dieses als nicht-trivialen Faktor zurück und terminiere.
- Andernfalls gilt $x \in \mathbb{Z}_n^*$.
- Bestimme mithilfe des Orakels (Shors Quantenalgorithmus) die Ordnung k von x modulo n.
- Das heißt, es gilt $x^k \equiv 1 \pmod{n}$ und $x^r \not\equiv 1 \pmod{n}$ für alle $r \in \{1, \ldots, k-1\}$.
- Wähle ein neues x, falls:
  - k ungerade ist, oder
  - $x^{k/2} \equiv -1 \pmod{n}$.
- Gib $\mathrm{ggT}(x^{k/2} - 1, n)$ als nicht-trivialen Faktor zurück.

Shor-Algorithmus: Beispiel

- Sei $n = 15$.
- x wird zufällig gewählt, z.B. $x = 2$.
- Es gilt: $\text{ggT}(x, n) = \text{ggT}(2, 15) = 1$.
- Das Orakel (Shors Quantenalgorithmus) liefert die Ordnung $k = 4$ von $x = 2$ modulo $n = 15$.
- Der Algorithmus berechnet:
  - $k/2 = 2$ is gerade.
  - $2^2 \mod 15 = 4 \neq -1 = 14$
- Also gibt der Algorithmus $\text{ggT}(2^2 - 1, 15) = 3$ zurück.
- Der andere Primteiler ist $\text{ggT}(x^{k/2} + 1,\ n) = \text{ggT}(5, 15) = 5$.
- IBM hat im Jahr 2001 einen Quantencomputer mit sieben Qubits eingesetzt, um die Zahl 15 in die Faktoren 3 und 5 zu zerlegen.

Shor-Algorithmus: Korrektheit

Behauptung: Der Shor-Algorithmus gibt einen nicht-trivialen Faktor von n aus.

Beweis:

- Betrachte das Produkt $(x^{k/2} - 1)(x^{k/2} + 1) = x^k - 1$.
- Es gilt $x^k - 1 \equiv 0 \pmod{n}$, d.h. n teilt $x^k - 1$
- Es gilt $(x^{k/2} - 1) \not\equiv 0 \pmod{n}$, weil k die kleinste Zahl mit $x^k - 1 \equiv 0 \pmod{n}$ ist. D.h. n teilt $(x^{k/2} - 1)$ nicht.
- Es gilt $(x^{k/2} + 1) \not\equiv 0 \pmod{n}$, weil x sonst neu gewählt würde. D.h. n teilt $(x^{k/2} + 1)$ nicht.
- Es folgt, dass $(x^{k/2} - 1)$ nicht-triviale Faktoren von n enthält.
- $\mathrm{ggT}(x^{k/2} - 1, n)$ liefert solch einen Faktor.

Man kann zeigen, dass der Shor-Algorithmus bei zufälliger Wahl von x mit Wahrscheinlichkeit $1/2$ einen der beiden Primteiler von n liefert, wenn n das Produkt von zwei verschiedenen ungeraden Primzahlen ist.

Rabin-Verfahren: Sicherheit

Das Problem, eine Quadratwurzel zu ziehen, modulo n (wobei $n = pq$), ist Orakel-reduzierbarkeitsäquivalent zum Faktorisieren.

Eine Richtung, nämlich Quadratwurzel berechnen $\preceq$ Faktorisieren, wurde bereits im Abschnitt über das Entschlüsseln im Rabin-Verfahren gezeigt.

Nun die andere Richtung, wieder probabilistisch:

- Wähle $x \in \mathbb{Z}_n^*$ zufällig (evtl. mehrfach).
- Berechne $y = x^2 \bmod n$.
- Das Orakel liefert eine Quadratwurzel z von y.
- Mit Wahrscheinlichkeit $1/2$ ist $z \not\equiv \pm x \pmod{n}$.
- Dann gilt $0 \equiv z^2 - x^2 \equiv (z - x)(z + x) \pmod{n}$.
- $\mathrm{ggT}(z - x, n)$ bzw. $\mathrm{ggT}(z + x, n)$ liefern die Primfaktoren von n.

Die Wiener Attacke auf RSA

- Um den Aufwand zur Entschlüsselung gering zu halten, könnte man einen kleinen Entschlüsselungsexponenten d wählen und den Verschlüsselungsexponenten e als multiplikativ inverses Element zu d wählen.
- $de \equiv 1 \pmod{\varphi(n)}$ bedeutet, dass es eine (unbekannte) Zahl $k \in \mathbb{N}$ gibt mit $de - k\varphi(n) = 1$.
- Wenn man beide Seiten der Gleichung durch $d\varphi(n)$ teilt, so ergibt sich

$$\frac{e}{\varphi(n)} - \frac{k}{d} = \frac{1}{d\varphi(n)}$$

- Die beiden Primfaktoren p und q von n sind ähnlich groß (beide sind ca. 500 Bit lang).
- $\varphi(n) = (p-1)(q-1) = n - p - q + 1$ ist ungefähr n.

Die Wiener Attacke auf RSA

- Also gilt
$$\frac{e}{n} - \frac{k}{d} \approx \frac{1}{dn}$$
- Wir haben eine (unbekannte) Approximation k/d zu dem bekannten Wert e/n mit ungefährem Fehler $1/dn$.
- Der Fehler $1/dn$ ist klein, verglichen mit $1/d$.
- Die Theorie der Kettenbrüche besagt, dass jede gute rationale Approximation mit kleinem Nenner (hier: d) mit dem Euklidischen Algorithmus berechnet werden kann.
- D.h. wenn wir den (leicht modifizierten) Euklidischen Algorithmus auf n und e anwenden, so wird d während des Verlaufs als Faktor auftauchen!
- Die Attacke geht zurück auf Wiener: Cryptanalysis of short RSA secret exponents. IEEE Trans. Inf. Theor. 36, 553–558 (1990).

Die Wiener Attacke auf RSA

- Wir modifizieren den Euklidischen Algorithmus so, dass er bei der Berechnung des größten gemeinsamen Teilers von zwei positiven ganzen Zahlen a und b auch die jeweiligen Quotienten ausgibt.
- Beispiel: Eingabe sei $a = 34$ und $b = 99$.

$$\begin{aligned} 34 &= 0 \cdot 99 + 34 \\ 99 &= 2 \cdot 34 + 31 \\ 34 &= 1 \cdot 31 + 3 \\ 31 &= 10 \cdot 3 + 1 \\ 3 &= 3 \cdot 1 + 0 \end{aligned}$$

Der ggT von 34 und 99 ist 1, das Tupel der Quotienten ist $(0, 2, 1, 10, 3)$.

Die Wiener Attacke auf RSA

- Seien a und b positive ganze Zahlen mit $\mathrm{ggT}(a, b) = 1$.
- Sei $(q_1, \ldots, q_m)$ das m-Tupel von Quotienten, dass der modifizierte Euklidische Algorithmus bei Anwendung auf a und b ausgibt.
- Dann gilt

$$\frac{a}{b} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{q_4 + \cdots + \cfrac{1}{q_m}}}}$$

- $(q_1, \ldots, q_m)$ heißt Kettenbruchentwicklung von $\frac{a}{b}$

Die Wiener Attacke auf RSA

- Für $1 \leq j \leq m$, definiere $C_j = (q_1, \ldots, q_j)$, d.h.

$$C_j = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{q_4 + \cdots + \cfrac{1}{q_j}}}}$$

- C_j heißt j-ter Näherungsbruch von $(q_1, \ldots, q_m)$.
- Im Beispiel mit dem Quotiententupel $(0, 2, 1, 10, 3)$ gilt

$$\begin{aligned}
C_1 &= 0 \\
C_2 &= 0 + \frac{1}{2} = \frac{1}{2} \\
C_3 &= 0 + \cfrac{1}{2 + \cfrac{1}{1}} = \frac{1}{3} \\
C_4 &= 0 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{10}}} = \frac{11}{32} \\
C_5 &= 0 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{10 + \cfrac{1}{3}}}} =
\end{aligned}$$

Die Wiener Attacke auf RSA

Jeder Näherungsbruch C_j kann als rationale Zahl $\frac{c_j}{d_j}$ geschrieben werden, wobei die c_j und d_j folgende Rekursionsgleichungen erfüllen:

$$c_j = \left\{ \begin{array}{ll} 1 & \text{if } j = 0 \\ q_1 & \text{if } j = 1 \\ q_j c_{j-1} + c_{j-2} & \text{if } j \geq 2 \end{array} \right\}$$

$$d_j = \left\{ \begin{array}{ll} 0 & \text{if } j = 0 \\ 1 & \text{if } j = 1 \\ q_j d_{j-1} + d_{j-2} & \text{if } j \geq 2 \end{array} \right\}$$

Die Berechnung der Näherungsbrüche C_j kann also in den Euklidischen Algorithmus integriert werden.

Die Wiener Attacke auf RSA

Theorem: Es gelte $\mathrm{ggT}(a, b) = 1$ und $\mathrm{ggT}(c, d) = 1$ sowie

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}$$

Dann ist $\frac{c}{d}$ einer der Näherungsbrüche der Kettenbruchentwicklung von $\frac{a}{b}$.

Beweis: von zur Gathen (Theorem 15.39)

Die Wiener Attacke auf RSA

Der RSA-Modul sei $n = p \cdot q$.

Satz: Wenn der Entschlüsselungsexponent d kleiner ist als $\frac{n^{\frac{1}{4}}}{3}$ und $q < p < 2q$ gilt, dann ist $\frac{k}{d}$ einer der Näherungsbrüche der Kettenbruchentwicklung von $\frac{e}{n}$.

Beweis: Wenn das Ergebnis der Berechnung von $\mathrm{ggT}(e, n)$ ungleich 1 ist, so ist e einer der Primfaktoren von n und wir können entschlüsseln. Ansonsten gilt $\mathrm{ggT}(e, n) = 1$.
Behauptung: Es gilt $\mathrm{ggT}(k, d) = 1$.
Beweis der Behauptung: Sei $\ell = \mathrm{ggT}(k, d)$. Aus $de - k\varphi(n) = 1$ folgt $\ell \cdot (\frac{d}{\ell}e - \frac{k}{\ell}\varphi(n)) = 1$ und damit $\ell = \pm 1$ (also die Beh.).
Man zeigt nun, dass $|\frac{e}{n} - \frac{k}{d}| < \frac{1}{3d^2}$ unter den gegebenen Voraussetzungen gilt (siehe Stinson 5.7.3) und wendet das vorherige Theorem an.

Die Wiener Attacke auf RSA: Beispiel

- Der öffentliche Schlüssel (n, e) bestehe aus $n = 160523347$ und $e = 60728973$.
- Die Kettenbruchentwicklung von $\frac{e}{n}$ ist

$$[0, 2, 1, 1, 1, 4, 12, 102, 1, 1, 2, 3, 2, 2, 36]$$

- Die ersten sechs Näherungsbrüche $C_j = \frac{c_j}{d_j}$ sind

$$0, \frac{1}{2}, \frac{1}{3}, \frac{2}{5}, \frac{3}{8}, \frac{14}{37}$$

Die Wiener Attacke auf RSA: Beispiel

- Woher weiß man, für welches j gilt: $d_j = d$?
- Wir wählen ein $m \in \mathbb{Z}_n^*$, z.B. $m = 2$, und überprüfen ob $(m^e)^{d_j} \equiv m \pmod{n}$ gilt.
- Für $1 \leq j \leq 5$ gilt $(2^{60728973})^{d_j} \equiv 94577066^{d_j} \not\equiv 2 \pmod{n}$.
- Für $j = 6$ gilt $(2^e)^{d_6} \equiv 94577066^{37} \equiv 2 \pmod{n}$.

Algorithmen zum Faktorisieren einer Zahl

Wir werden die folgenden Faktorisierungsalgorithmen betrachten:

- $(p-1)$-Methode nach Pollard zur Faktorisierung
- Pollard-ρ für das Faktorisieren
- Fermat-Faktorisierung
- Diese sind mit naiven Algorithmen zu vergleichen.
- Ein naiver Faktorisierungsalgorithmus würde bei Eingabe n alle Probeteiler bis $\sqrt{n}$ durchprobieren.
- Hierzu veranschlagen wir die Laufzeit $O(\sqrt{n}) = O(\sqrt{2^k}) = O(2^{k/2}) \approx O(1.414^k)$, wobei k die Bitlänge von n ist.
- (Wir ignorieren hier evtl. weitere polynomiale Terme, da diese durch den Exponentialterm bei weitem majorisiert werden.)

Fermat-Faktorisierung

- c't Magazin vom 01.06.2022: Schwache RSA-Schlüssel mit 380 Jahre altem Faktorisierungsalgorithmus knacken
- https://www.heise.de/hintergrund/Schwache-RSA-Schluessel-mit-380-Jahre-altem-Faktorisierungsalgorithmus-knacken-7096427.html
- Drucker von Canon und Fujifilm benutzten einen RSA-Modul, der mit dem Fermat-Faktorisierung-Algorithmus faktorisiert werden konnte.
- Es geht also um die Primfaktorzerlegung des RSA-Moduls n.

Fermat-Faktorisierung

Satz: Jede ungerade, zusammengesetzte ganze Zahl $n > 9$ läßt sich als Differenz zweier Quadratzahlen $n = x^2 - y^2$ mit $x \geq \sqrt{n}$ schreiben.

Beweis. Sei $n = p \cdot q$ mit $p > q$. Sowohl p als auch q müssen ungerade Zahlen sein. Mit $x = (p + q)/2$ und $y = (p - q)/2$ gilt

$$x^2 - y^2 = (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2 = p \cdot q = n$$

D.h. n lässt sich in $n = p \cdot q$ faktorisieren genau dann, wenn sich n als Differenz zweier Quadratzahlen schreiben lässt, $n = x^2 - y^2$.

Fermat-Faktorisierung

Der Fermatsche Faktorisierungs-Algorithmus arbeitet nun wie folgt:

- Sei $x_0 = \lceil \sqrt{n} \rceil$ die kleinste ganze Zahl $\geq \sqrt{n}$.
- Für $d \in \mathbb{N}$ berechnet man die Differenzen $d_k = x_k^2 - n$, wobei $x_k = x_0 + k$, bis man auf eine Quadratzahl $d_k = y^2$ stößt.
- Dann ist $n = (x_k + y)(x_k - y)$.
- Bei der sukzessiven Berechnung der d_k kann man folgende Eigenschaft benutzen:

$$d_{k+1} - d_k = x_{k+1}^2 - x_k^2 = (x_k + 1)^2 - x_k^2 = 2x_k + 1$$

Fermat-Faktorisierung: Beispiel

- Sei $n = 6161$.
- Dann ist $x_0 = \lceil \sqrt{6161} \rceil = 79$.
- $d_0 = 79^2 - 6161 = 80$ (keine Quadratzahl).
- $d_1 = 80^2 - 6161$ wird berechnet durch
- $d_1 = d_0 + (2x_0 + 1) = 80 + (2 \cdot 79 + 1) = 239$ (keine Quadratzahl).
- $d_1 = 81^2 - 6161$ wird berechnet durch
- $d_2 = d_1 + (2x_1 + 1) = 239 + (2 \cdot 80 + 1) = 400 = 20^2$
- Also ergibt sich $p = 81 + 20 = 101$ und $q = 81 - 20 = 61$.

Fermat-Faktorisierung

- Effizient ist der Algorithmus nur, falls p und q sehr nahe beieinander liegen.
- Die x-Werte müssen die Zahlen von $\lceil\sqrt{n}\rceil = \lceil\sqrt{pq}\rceil$ bis $(p+q)/2$ durchlaufen.
- Nimmt man Größenordnungen $n \approx 2^{1000}$, $p \approx 2^{499}$, $q \approx 2^{501}$ wie sie bei RSA realistisch sind, so ist die Anzahl der Schleifendurchgänge

$$(p+q)/2 - \lceil\sqrt{n}\rceil \approx \left(\frac{1}{2}\cdot 2^{500} + 2\cdot 2^{500}\right)/2 - 2^{500} = \frac{2^{500}}{4} = 2^{498}$$

  also etwa wie beim naiven Algorithmus.
- Allgemeiner: die Laufzeit im worst case ist $O(\sqrt{n}) = O(2^{k/2})$.
- Die Idee der Fermat-Faktorisierung ist aber Grundlage für viele raffiniertere Faktorisierungsalgorithmen.

Verallgemeinerung der Fermat-Faktorisierung

Die Grundidee der Fermat-Faktorisierung-Methode und weiterer, daraus entwickelter Methoden, kann man folgendermaßen formulieren:

- Man sucht nach ganzen Zahlen x, y mit $x^2 \equiv y^2 \pmod{n}$.
- Das bedeutet $(x - y) \cdot (x + y) = kn$ für ein k.
- Sofern $x \not\equiv \pm y \pmod{n}$, so ergibt sich mittels $ggT(x + y, n)$ bzw. $ggT(x - y, n)$ ein nicht-trivialer Teiler von n
- Dixons Faktorisierungsmethode beruht auf dieser Idee.
- Das Quadratische Sieb ist eine Weiterentwicklung von Dixons Faktorisierungsmethode.
- Das Zahlkörpersieb wiederum kann als Verallgemeinerung des Quadratischen Siebes verstanden werden.

Pollards $(p-1)$-Methode

- Diese Methode ist besonders effizient, wenn für einen der Faktoren von n, z.B. p, gilt, dass $p-1$ in ausschließlich kleine Primfaktoren zerfällt.
- D.h. $p-1$ ist Teiler von $B!$ für eine nicht zu große Zahl B.
- Sei beispielsweise $p = 2161$ Primzahl.
- Dann ist $p - 1 = 2160 = 2^4 \cdot 3^3 \cdot 5$.
- Dann ist $p-1$ Teiler von

$$9! = 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 = 2^7 \cdot 3^4 \cdot 5 \cdot 7$$

- In diesem Fall gilt mit Hilfe des Satzes von Fermat:

$$a := 2^{B!} = 2^{(p-1)\cdot t} \equiv 1^t = 1 \pmod{p}$$

- Somit ist $a-1$ Vielfaches von p. Die Berechnung von $ggT(a-1, n)$ ergibt dann den Faktor p.

Pollards $(p-1)$-Methode

- Der Algorithmus testet der Reihe nach $B = 1, 2, 3, \ldots$.
- Die Berechnung von $2^{B!}$ geschieht iterativ mit der Formel $2^{B!} = \left(2^{(B-1)!}\right)^B$.

```
PROC pollardpminus1(n);
a := 2;
B := 1;
REPEAT
    B := B + 1;
    a := modexp(a, B, n);
    g := ggT(a − 1, n);
UNTIL g ≠ 1;
IF g = n THEN
    RETURN "failure"
ELSE
    RETURN g;
```

Pollards $(p-1)$-Methode

Bemerkung: Eine kleine Chance besteht, dass der ggT die Zahl n ergibt, dann versucht man es mit einer anderen Basiszahl als 2.

Der worst case liegt vor, wenn

1. $p-1 = 2 \cdot p'$ und $q-1 = 2 \cdot q'$ für zwei Primzahlen p', q', also wenn p und q sichere Primzahlen sind,
2. p' und q' von ähnlicher Größenordnung sind.

- Im worst case gilt $p' \in O(\sqrt{n})$ und $q' \in O(\sqrt{n})$.
- Die worst case Zeitkomplexität von Pollards $(p-1)$-Methode ist damit $O(\sqrt{n}) = O(2^{k/2})$.

Pollards $(p-1)$-Methode: Beispiel

- Sei $n = 4163 = 181 \cdot 23$.
- Es gilt $180 = 2^2 \cdot 3^2 \cdot 5$, also ist 180 Teiler von 6!.
- 23 ist eine sichere Primzahl, es gilt $22 = 2 \cdot 11$.
- 22 ist also ein Teiler von 11!.
- Deshalb wird zuerst der Primfaktor 181 gefunden.
- Wir verfolgen den Verlauf der Berechnung:

B	$=$	2	3	4	5	6
a	$=$	4	64	326	3209	906
g	$=$	1	1	1	1	181

Erinnerung: Geburtstagsproblem

- Wie viele Personen müssen zusammenkommen, um eine Wahrscheinlichkeit $\geq 1/2$ zu erreichen, dass mindestens zwei Personen am selben Tag im Jahr Geburtstag haben?
- Annahme: die Geburtstage der Personen sind unabhängige, identisch verteilte Zufallsvariablen aus der diskreten Gleichverteilung auf der 365-elementigen Menge der Tage.
- Die Antwort auf die Frage lautet: 23 Personen.
- Im Urnenmodell entspricht obige Annahme einer Ziehung von Kugeln mit Zurücklegen aus einer Urne, die $m = 365$ verschiedene Kugeln enthält.
- Theorem: Die erwartete Anzahl von Ziehungen, bis eine Kugel zum zweiten Mal auftaucht, ist $\approx 1,25\sqrt{m}$, also $O(\sqrt{m})$.

Pollard-ρ-Algorithmus

- Gegeben sei die Eingabe $n = p \cdot q$, wobei p kleinster Primfaktor von n ist. Dann gilt $p \leq \sqrt{n}$.
- Man erzeugt (Pseudo-)Zufallszahlen $z_1, z_2, \cdots < n$.
- Hierbei hat sich folgender Pseudo-Zufallszahlengenerator als günstig herausgestellt: Nach Wahl einer beliebigen Anfangszahl z_0 rechnet man $z_{i+1} = f(z_i) = (z_i^2 + 1) \bmod n$.
- Wegen des Geburtstagsparadoxons wird es nach ungefähr $\sqrt{p} \leq n^{1/4}$ solchen Zufallszahlen $i < j$ geben mit

$$z_i \equiv z_j \pmod{p}$$

- Es ist sehr wahrscheinlich, dass dann $z_i \not\equiv z_j \pmod{q}$ und damit $z_i \neq z_j$ gilt.
- Wenn $z_i \neq z_j$, so liefert $ggT(z_i - z_j, n)$ einen Teiler von n, nämlich p, denn $z_i - z_j = k \cdot p$ für eine ganze Zahl k.
- Nachteil der Methode: Man muss alle Werte speichern bis eine Kollision auftritt.

Pollard-ρ-Algorithmus

Behauptung: Da Pseudozufallszahlen verwendet werden, folgt (modulo p) aus $z_i \equiv z_j$ auch $z_{i+1} \equiv z_{j+1}$, $z_{i+2} \equiv z_{j+2}$, usw.

Beweis:

- $z_i \equiv z_j \pmod{p}$ impliziert $(z_i^2 + 1) \equiv (z_j^2 + 1) \pmod{p}$.
- Erinnerung: $z_{i+1} = (z_i^2 + 1) \bmod n$.
- $z_{i+1} \bmod p = (z_i^2 + 1 \bmod n) \bmod p = z_i^2 + 1 \bmod p$, weil p Teiler von n ist.
- Analog gilt: $z_{j+1} \bmod p = z_j^2 + 1 \bmod p$.
- Es folgt: $z_{i+1} \equiv z_{j+1} \pmod{p}$.
- Die Behauptung folgt durch wiederholtes Anwenden des Argumentes.

Pollard-ρ-Algorithmus

Cycle Detection Trick: Anstatt alle (i, j)-Kombinationen zu überprüfen, betrachte nur $(k, 2k)$ für $k = 1, 2, \ldots,$ und teste, ob $ggT(z_k - z_{2k}, n)$ einen nicht-trivialen Teiler ergibt.

```
PROC Pollardrho(n);
x := random(1, n − 1);
y := x;
REPEAT
    x := (x^2 + 1) mod n;
    y := (y^2 + 1) mod n;
    y := (y^2 + 1) mod n;
    g := ggT(x − y, n);
UNTIL g ≠ 1;
IF g = n THEN
    RETURN "failure"
ELSE
    RETURN g;
```

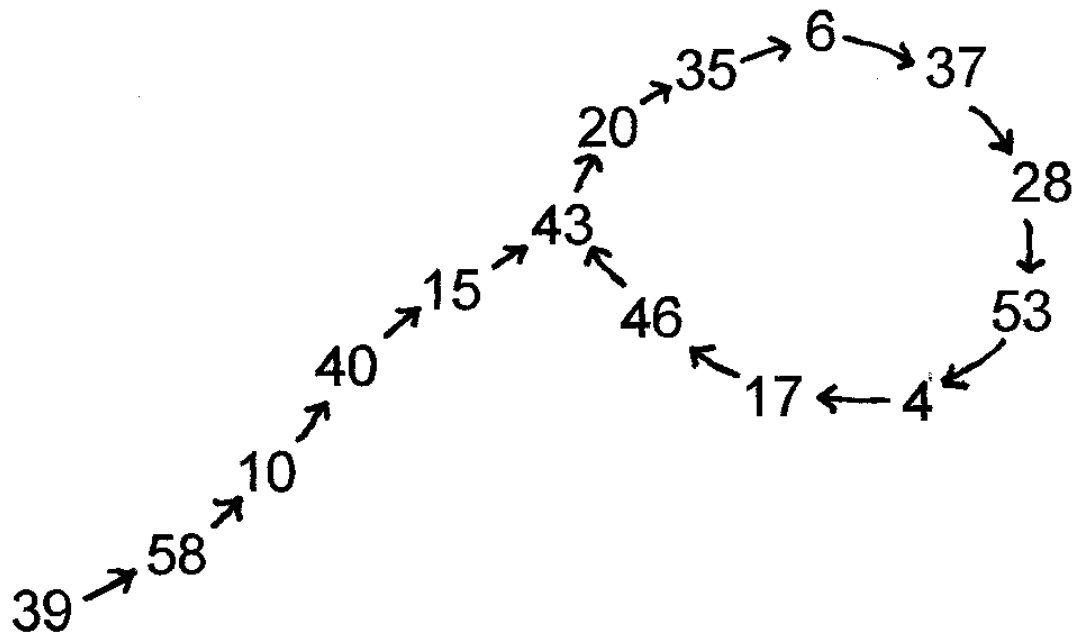
Pollard-ρ-Algorithmus: Beispiel

- Sei $n = 6161 = 61 \cdot 101$.
- Die erste Pseudozufallszahl sei 100.
- Dann ergeben sich die ersten 20 Pseudozufallszahlen als $(z_1, \ldots, z_{20}) = (100, 3840, 2328, 4066, 2394, 1507, 3802, 1499, 4398, 3026, 1431, 2310, 675, 5873, 2852, 1385, 2155, 4793, 4642, 3148)$.
- Bei $k = 10$ ergibt sich der Treffer:

$$ggT(z_{10} - z_{20}, n) = ggT(3026 - 3148, 6161) = 61$$

Pollard-ρ-Algorithmus: Beispiel

Betrachten wir diese Pseudozufallsfolge modulo 61, so ergeben sich folgende Zahlen (die Form dieses Diagramms gibt der Methode ihren Namen nach dem griechischen Buchstaben rho):



Man sieht, dass der erste Treffer bereits bei $i = 6, j = 16$ vorliegt, denn $z_6 \equiv z_{16} \equiv 43 \pmod{61}$, danach bei $i = 7, j = 17$, usw. bis der Cycle Detection Trick den Treffer bei $k = 10, 2k = 20$ findet.

Pollard-ρ-Algorithmus

- Cycle Detection Trick: Anstatt alle (i, j)-Kombinationen zu überprüfen, betrachte nur $(k, 2k)$ für $k = 1, 2, \ldots,$ und teste, ob $ggT(z_k - z_{2k}, n)$ einen nicht-trivialen Teiler ergibt.
- Behauptung: Wenn (i, j), $i < j$, das erste Paar mit $z_i \equiv z_j \pmod{p}$ ist, so findet man mit dem Cycle Detection Trick ein Paar $(k, 2k)$ mit $i \leq k < j$ und $z_k \equiv z_{2k} \pmod{p}$.
- Beweis: Sei $\ell = j - i$ die Länge der Schleife. Aus der Annahme $z_i \equiv z_j \pmod{p}$ folgt $z_k \equiv z_{2k} \pmod{p}$ für alle k mit $k \equiv 0 \pmod{\ell}$ und $k \geq i$. Unter den ℓ aufeinanderfolgenden Zahlen $i, \ldots, j-1$ muss es eine Zahl geben, die durch ℓ teilbar ist. Diese Zahl erfüllt die Bedingungen.
- Folgerung: Die Anzahl der benötigten Iterationen, um einen Faktor p zu finden, ist höchstens j; d.h. der Erwartungswert ist auch mit dem Cycle Detection Trick $O(\sqrt{p})$.

Pollard-ρ-Algorithmus

- Die Laufzeit ist also (im Erwartungswert):
  $O(\sqrt{p}) = O(n^{1/4}) = O(2^{k/4}) \approx O(1.189^k)$,
  wobei $k =$ Bitlänge von n.
- Man beachte die seltsame Ambivalenz bei der Verwendung von Pseudozufallszahlen.
- Einerseits sollen diese idealerweise wie echte Zufallszahlen sein (damit das Geburtstagsphänomen eintritt).
- Andererseits verwendet man gerade beim Cycle Detection Trick die Tatsache, dass dies Pseudozufallszahlen sind, die durch eine deterministische Berechnung auseinander hervorgehen, was bei echten Zufallszahlen nicht funktionieren würde.

Algorithmen zur Berechnung des Diskreten Logarithmus

- Sei n eine Primzahl.
- Sei a eine Primitivwurzel modulo n, d.h. a erzeugt $\mathbb{Z}_n^*$ (insbesondere ist $\varphi(n) = n - 1$ die Ordnung von a).
- Dann hat jede Zahl $y \in \mathbb{Z}_n^*$ einen eindeutigen Exponenten $x \in \{0, 1, 2, \ldots, n-2\}$ mit $y = a^x \mod n$.
- Dieser Exponent x ist der *diskrete Logarithmus* von y (zur Basis a, modulo n).
- Aufgabe: Bestimme den diskreten Logarithmus x von $y \in \mathbb{Z}_n^*$.
- Ein naiver Algorithmus würde alle $x < n$ ausprobieren, bis sich einmal $y = a^x \mod n$ ergibt. Der Algorithmus hat die Laufzeit $O(n) = O(2^k)$, wobei k wieder die Bitlänge ist.

Wir werden im Folgenden zwei Algorithmen zur Berechnung des Diskreten Logarithmus besprechen:

- Babystep-Giantstep-Algorithmus
- Pollard-ρ-Algorithmus

Babystep-Giantstep-Algorithmus

- Man kann $y = a^x$ umschreiben in (alle Berechnungen sind modulo n zu verstehen)

$$y = a^x = a^{x_1 \cdot \lceil \sqrt{n} \rceil + x_2} = \left(a^{\lceil \sqrt{n} \rceil}\right)^{x_1} \cdot a^{x_2} \quad \text{mit} \quad x_1, x_2 \leq \lfloor \sqrt{n} \rfloor$$

- Dabei ist x_2 der Rest und x_1 der Quotient der Division von x durch $\lceil \sqrt{n} \rceil$.
- Nun geht es darum, x_1, x_2 zu bestimmen, was mit geringerem Aufwand geht, als alle x durchzuprobieren.
- Umgeformt erhalten wir:

$$y \cdot (a^{-1})^{x_2} = \left(a^{\lceil \sqrt{n} \rceil}\right)^{x_1}$$

- Wir kürzen a^{-1} mit u ab, und wir kürzen $a^{\lceil \sqrt{n} \rceil}$ mit v ab.
- Gesucht sind x_1, x_2, so dass $y \cdot u^{x_2} = v^{x_1}$.
- Berechne die Liste $L = \{ (x_1, v^{x_1}) \mid x_1 = 0, 1, \ldots, \lfloor \sqrt{n} \rfloor \}$.

Babystep-Giantstep-Algorithmus

- Die Liste $L = \{ (x_1, v^{x_1}) \mid x_1 = 0, 1, \ldots, \lfloor \sqrt{n} \rfloor \}$ der so genannten Giantsteps benötigt $O(\sqrt{n})$ Speicherplatz.
- Wir verwenden eine effiziente Datenstruktur (Suchbaum, Hashtabelle), so dass bei Anfrage von z effizient festgestellt werden kann, ob $(x_1, z) \in L$ gilt für ein x_1.
- Wenn ja, so wird dieses x_1 ausgegeben.
- Der folgende Algorithmus mit $O(\sqrt{n}) = O(2^{k/2}) \approx O(1.414^k)$ Laufzeit berechnet alle Babysteps.
- Wenn ein Babystep $y \cdot u^{x_2}$ mit einem Giantstep v^{x_1} übereinstimmt, so wird $(x_1 \cdot \lceil \sqrt{n} \rceil + x_2)$ als diskreter Logarithmus von y (zur Basis a, modulo n) ausgegeben.

```
FOR x2 := 0 TO ⌊√n⌋ DO
    IF Anfrage y · u^x2 an Liste L liefert das Ergebnis x1
        THEN OUT( x1 · ⌈√n⌉ + x2 );
```

Babystep-Giantstep-Algorithmus: Beispiel

- Sei $n = 227$ (Primzahl). $a = 5$ ist Primitivwurzel mod 227.
- Es soll der diskrete Logarithmus von $y = 86$ berechnet werden.
- Wir berechnen $u = 5^{-1} = 91$ und $v = 5^{\lceil\sqrt{n}\rceil} = 5^{16} = 175$.
- Dann ergibt sich folgende Liste: $L = \{(0,1),(1,175),$

$$(2,207),(3,132),(4,173),(5,84),(6,172),(7,136),(8,192),$$

$$(9,4),(10,19),(11,147),(12,74),(13,11),(14,109),(15,7)\}$$

- Die FOR-Schleife liefert einen Treffer bei der Anfrage, ob $y \cdot u^{x_2} = 86 \cdot 91^9 = 192$ in der Liste vorkommt.
- Die positive Antwort ergibt $x_1 = 8$.
- Damit ergibt sich die Lösung

$$x = \lceil\sqrt{n}\rceil \cdot x_1 + x_2 = 16 \cdot 8 + 9 = 137$$

- Probe: $5^{137} \bmod 227 = 86$

Pollard-ρ-Algorithmus

- Gegeben: Primzahl n, Primitivwurzel a und $y < n$.
- Aufgabe: Finde x so dass $y = a^x \bmod n$.
- Mittels (Pseudo-) Zufallsgenerator erzeuge $z_1 = a^{i_1} y^{j_1}$, $z_2 = a^{i_2} y^{j_2}, \ldots$ (alles mod n).
- Wegen des Geburtstagsparadoxons gibt es nach ungefähr $\sqrt{n}$ Schritten $\mu < \sigma$ mit $z_\mu = z_\sigma$, also $a^{i_\mu} y^{j_\mu} \equiv a^{i_\sigma} y^{j_\sigma} \pmod{n}$.
- Somit gilt $a^{i_\mu} a^{x \cdot j_\mu} \equiv a^{i_\sigma} a^{x \cdot j_\sigma} \pmod{n}$ und damit

$$i_\mu + x \cdot j_\mu \equiv i_\sigma + x \cdot j_\sigma \pmod{n-1}$$

- Die Kongruenzgleichung muss man nach x auflösen.

Pollard-ρ-Algorithmus

- Diesmal muss der Pseudozufallsgenerator f so beschaffen sein, dass aus $z_\mu = z_\sigma$ folgt $f(z_\mu) = f(z_\sigma)$.
- Gleichzeitig müssen dabei die Exponenten i, j im Term $a^i y^j$ verändert werden.
- Dies kann man folgendermaßen erreichen: Zerlege die Menge $M = \{1, \ldots, n-1\}$ in drei etwa gleichgroße Teilmengen:

$$M = M_1 \dot\cup M_2 \dot\cup M_3$$

- Gegeben sei $z = a^i \cdot y^j \bmod n$, z.B. $(z, i, j) = (1, 0, 0)$.
- Dann ergibt sich die darauf folgende Pseudozufallszahl $z' = a^{i'} \cdot y^{j'}$ mithilfe von

$$(z', i', j') = \left\{ \begin{array}{ll} (a \cdot z \bmod n, (i+1) \bmod (n-1), j) & \text{if } z \in M_1 \\ (z^2 \bmod n, 2i \bmod (n-1), 2j \bmod (n-1)) & \text{if } z \in M_2 \\ (z \cdot y \bmod n, i, (j+1) \bmod (n-1)) & \text{if } z \in M_3 \end{array} \right\}$$

Pollard-ρ-Algorithmus: Beispiel

- 97 ist Primzahl und 17 ist eine Primitivwurzel.
- Wir suchen x so dass $10 = 17^x$. (Die Rechnungen sind modulo 97 zu verstehen.)
- Wie vorgesehen zerlegen wir in 3 Mengen: $M_1 = \{1, \ldots, 32\}$, $M_2 = \{33, \ldots, 64\}$, $M_3 = \{65, \ldots, 96\}$.
- Wir starten willkürlich mit der Zahl $a^0 y^{10}$.
- Dies ergibt folgende Zahlenfolge:

$$a^0 y^{10} = 49 \in M_2,\ a^0 y^{20} = 73 \in M_3,\ a^0 y^{21} = 51 \in M_2,$$

$$a^0 y^{42} = 79 \in M_3,\ a^0 y^{43} = 14 \in M_1,\ a^1 y^{43} = 44 \in M_2,$$

$$a^2 y^{86} = 93 \in M_3,\ a^2 y^{87} = 57 \in M_2,\ a^4 y^{78} = 48 \in M_2,\ a^8 y^{60} = 73$$

- Bei den Exponenten muss modulo 96 gerechnet werden.
- Es gilt: $a^8 y^{60} = a^0 y^{20} = 73$, und von nun ab wiederholen sich die Zahlen ab der 73 zyklisch.

Pollard-ρ-Algorithmus: Beispiel

- Es gilt: $a^8(a^x)^{60} = a^8 y^{60} = 73 = a^0 y^{20} = a^0(a^x)^{20}$
- D.h. $a^{8+60x} = a^{20x}$.
- Wir müssen die Kongruenz $40x \equiv -8 \pmod{96}$ lösen.

Es gilt folgender Satz:

1. Die lineare Kongruenz $ax \equiv b \pmod{m}$ ist lösbar genau dann, wenn $\mathrm{ggT}(a, m) | b$.
2. Gilt $\mathrm{ggT}(a, m) | b$, dann ist die Anzahl der paarweise inkongruenten Lösungen gleich $\mathrm{ggT}(a, m)$, d.h. für teilerfremde Zahlen a und m ist die lineare Kongruenz eindeutig lösbar.

Ist x_0 eine spezielle Lösung, so ist $\{x_0 + \frac{m}{ggT(a,m)} \cdot k \mid k \in \mathbb{Z}\}$ die Menge aller Lösungen.

Pollard-ρ-Algorithmus: Beispiel

- Wir müssen die Kongruenz $40x \equiv -8 \pmod{96}$ lösen.
- Es gilt $d := \mathrm{ggT}(40, 96) = 8$.
- Wir dividieren die Kongruenzgleichung durch $d = 8$ und erhalten $5x \equiv -1 \pmod{12}$.
- Da nun $\mathrm{ggT}(5, 12) = 1$ gilt, ist 5 in $\mathbb{Z}_{12}^*$ invertierbar.
- Das multiplikativ Inverse von 5 in $\mathbb{Z}_{12}^*$ ist wiederum 5.
- Es folgt $x \equiv -5 \pmod{12}$.
- Damit ist $x = 7$ eine Lösung.

Pollard-ρ-Algorithmus: Beispiel

- Wir müssen die Kongruenz $40x \equiv -8 \pmod{96}$ lösen.
- Eine spezielle Lösung ist $x_0 = 7$, denn

$$7 \cdot 40 + -3 \cdot 96 = 280 - 288 = -8$$

- Es gilt $d = \text{ggT}(40, 96) = 8$ und $m/d = 96/8 = 12$.
- Die Menge der 8 inkongruenten Lösungen ist damit

$$M = \{7 + 12k \mid 0 \leq k \leq 7\} = \{7, 19, 31, 43, 55, 67, 79, 91\}$$

- Man überprüft nun, für welches $x \in M$ gilt

$$17^x \equiv 10 \pmod{97}$$

- Die Lösung lautet $x = 91$.

Pollard-ρ-Algorithmus

- Ein Nachteil der obigen Methode ist, dass alle Werte gespeichert werden müssen, bis eine Kollision auftritt.
- Wie bei beim Pollard-ρ-Algorithmus zum Faktorisieren, kann man den Cycle Detection Trick anwenden.
- D.h. man durchläuft die Indexfolge $(i, 2i)$, $i = 1, 2, \ldots,$ bis man bis man $z_i = z_{2i}$ erhält.
- So wird nur konstant viel Speicherplatz verwendet.
- Die erwartete Laufzeit ist

$$O(\sqrt{n}) = O(n^{1/2}) = O(2^{k/2}) \approx O(1.414^k)$$

wobei $k =$ die Bitlänge von n ist.

Diskreter Logarithmus: Sicherheit

- Erinnerung: Sowohl der Diffie-Hellman-Schlüsselaustausch als auch das ElGamal-Verfahren arbeiten ursprünglich auf der zyklischen Gruppe $\mathbb{Z}_p^*$, wobei p eine große Primzahl ist.
- Zur Berechnung von diskreten Logarithmen in $\mathbb{Z}_p^*$ kann außer dem Babystep-Giantstep-Algorithmus und dem ρ-Algorithmus von Pollard auch der Index-Calculus-Algorithmus (wird hier nicht behandelt) benutzt werden.
- Das Sicherheitsniveau 80 (2^{80} Möglichkeiten ausprobieren) erfordert Primzahlen der Größe 1024 Bit.
- Sowohl der Diffie-Hellman-Schlüsselaustausch als auch das ElGamal-Verfahren können in anderen Gruppen als $\mathbb{Z}_p^*$ realisiert werden.

Elliptische-Kurven-Kryptografie

- Unter Elliptische-Kurven-Kryptografie versteht man asymmetrische Kryptosysteme, die Operationen auf elliptischen Kurven über endlichen Körpern verwenden.
- Auf elliptischen Kurven kann eine additive zyklische Gruppe definiert werden, die aus den Vielfachen eines Punktes auf der Kurve, des Erzeugers der Gruppe, besteht.
- Das Problem des diskreten Logarithmus in elliptischen Kurven ist deutlich schwerer zu lösen als in der Gruppe $\mathbb{Z}_p^*$, weil der Index-Calculus-Algorithmus in diesem Fall nicht angewendet werden kann.
- Nach heutigem Kenntnisstand wird z.B. mit einer Schlüssellänge von 160 Bit eine ähnliche Sicherheit erreicht wie bei Diffie-Hellman mit $\mathbb{Z}_p^*$ (oder RSA) mit 1024 Bit.
- Elliptische-Kurven-Kryptografie eignet sich daher besonders dann, wenn die Speicher- oder Rechenkapazität begrenzt ist, z.B. in Smartcards oder anderen eingebetteten Systemen.

Digitale Signaturen

Eine elektronische *Unterschrift* bedeutet, dass einem Dokument m ein Text u hinzugefügt wird, also (m, u), der als Unterschrift des Teilnehmers A zu verstehen ist.

Dazu müssen verschiedene Bedingungen gelten:

- Für jeden soll überprüfbar sein, dass u die Unterschrift von A an das Dokument m ist und nicht von einer anderen Person stammt.
- Nur A soll es möglich sein, seine Unterschrift herzustellen. Keiner anderen Person soll es möglich sein As Unterschrift zu fälschen. Insofern kann A auch nicht abstreiten, das Dokument unterschrieben zu haben.
- Das unterschriebene Dokument (m, u) kann nicht in einer Weise abgefälscht werden zu $(\tilde{m}, \tilde{u})$, so dass dies als As Unterschrift unter ein anderes Dokument $\tilde{m}$ angesehen werden kann.

Digitale Signaturen

- In einer Public Key-Infrastruktur hat jeder Teilnehmer A einen öffentlichen Schlüssel k_A und einen privaten, also geheimen Schlüssel k'_A.
- Es gibt öffentlich bekannte Ver- und Entschlüsselungsfunktionen E und D. In diesem Szenario kann A eine Unterschrift an ein Dokument m dadurch herstellen, dass er $u = D(k'_A, m)$ berechnet.
- Nur A, der im Besitz von k'_A ist, kann diese Information herstellen.
- Von jedem kann überprüft werden, dass u tatsächlich die Unterschrift von A an das Dokument m ist, indem man $m' = E(u, k_A) = E(D(k'_A, m), k_A)$ berechnet und überprüft, dass $m = m'$ gilt.
- Da E (in Kombination mit k_A) und D (in Kombination mit k'_A) sich gegenseitig aufheben (also die identische Abbildung ergeben), sollte $m = m'$ sein.

Digitale Signaturen

- In Kryptosystemen wird beim Verschlüsseln von Nachrichten zuerst E auf m angewandt und beim Entschlüsseln D auf die Chiffre c. Das Ergebnis beider Anwendungen ist m.
- Bei Signaturen soll gelten: wenn man zuerst D auf m anwendet und dann E auf die Unterschrift, soll ebenfalls am Ende m herauskommen.
- Das gilt nur, falls die Funktionen E und D solcherart sind, dass sie miteinander kommutieren, also die Reihenfolge der Anwendung vertauscht werden kann.
- Bei RSA ist die Kommutativität vom E und D gegeben, bei ElGamal ist sie jedoch nicht gegeben.

RSA-Signaturen

- Bei RSA ist dies trivialerweise der Fall, da beide Funktionen die modulare Exponentiation darstellen und da die Potenzen e und d im Exponenten aufgrund der Kommutativität der Multiplikation vertauscht werden dürfen:

$$(m^d)^e = m^{de} = m^{ed} = (m^e)^d$$

- Unterschreiben von m mittels RSA bedeutet: $u = m^d \bmod n$ herstellen.
- Verifizieren, dass u tatsächlich As Unterschrift ist, geschieht durch den Test, ob $m = u^e \bmod n$ gilt.
- Hierbei ist (n, e) der öffentliche RSA-Schlüssel von A und d der geheime.

RSA-Signaturen: Beispiel

- Bobs öffentliche Schlüssel seien $n = 33$ und $e = 3$.
- Geheim bleibt $d = 7$.
- Möchte man sich das Dokument $m = 15$ von Bob unterschreiben lassen, so berechnet dieser $(m^d \bmod n) = \mathit{modexp}(15, 7, 33) = 27$.
- Das Dokument, zusammen mit Bobs Unterschrift, ist dann $(15, 27)$.
- Bobs Unterschrift überprüfen kann man durch den Test $m \stackrel{?}{=} (27^3 \bmod 33)$.
- Beide Seiten der Gleichung ergeben 15.

Erinnerung: ElGamal-Verfahren

- Öffentliche Informationen: eine große Primzahl n und eine Primitivwurzel a modulo n.
- Bob wählt eine geheime Zufallszahl $y < n$ und berechnet den öffentlichen Schlüssel $\tilde{y} = a^y \bmod n$.
- Alice wählt eine Zufallszahl $x < n$ und berechnet $\tilde{x} = a^x \bmod n$ sowie $z = \tilde{y}^x \bmod n$.
- Sie verschlüsselt die Nachricht m zu $c = (m \cdot z) \bmod n$.
- Alice sendet die Information $(\tilde{x}, c)$ an Bob.
- Bob berechnet $\tilde{x}^y \bmod n$.
- Die Berechnung ergibt die Zufallszahl z, denn modulo n gilt: $\tilde{x}^y = a^{xy} = \tilde{y}^x$.
- Er entschlüsselt dann mittels $(c \cdot z^{-1}) \bmod n$ und erhält so m.

ElGamal-Signatur: Konstruktion der Unterschrift

- Sei n Primzahl und a eine Primitivwurzel modulo n.
- Nehmen wir an, ein Teilnehmer möchte ein Dokument m unterschreiben.
- Initial stellt der Teilnehmer den öffentlichen Schlüssel $\tilde{k}$ und den geheimen Schlüssel k her, wobei $\tilde{k} = a^k \bmod n$.
- Nun erzeugt dieser eine Zufallszahl z (teilerfremd zu $n-1$) und berechnet $\tilde{z} = a^z \bmod n$.
- Ferner löst der Teilnehmer die modulare Gleichung $m \equiv k \cdot \tilde{z} + z \cdot x \pmod{(n-1)}$ nach x auf.
- Dies ist dadurch möglich, dass mithilfe des Erweiterten Euklid-Algorithmus die Zahl $z^{-1} \bmod (n-1)$ bestimmt wird.
- Dann ergibt sich $x = ((m - k \cdot \tilde{z}) \cdot z^{-1}) \bmod (n-1)$.
- Die Unterschrift besteht nun aus dem Paar $(\tilde{z}, x)$.

ElGamal-Signatur: Verifikation

- Dokument samt Unterschrift sei das Paar $(m, (\tilde{z}, x))$.
- Die Unterschrift kann überprüft werden, indem man verifiziert, dass $a^m \bmod n$ und $(\tilde{k})^{\tilde{z}} \cdot (\tilde{z})^x \bmod n$ identisch sind.
- Man beachte, dass alle beteiligten Zahlen $a, n, \tilde{k}, \tilde{z}, x, m$ öffentlich sind, also für den Verifikationsvorgang zur Verfügung stehen.
- Die Verifikation ist korrekt, denn

$$a^m \equiv a^{k \cdot \tilde{z} + z \cdot x} \equiv (a^k)^{\tilde{z}} \cdot (a^z)^x \equiv (\tilde{k})^{\tilde{z}} \cdot (\tilde{z})^x \pmod{n}$$

- Es ist nur für den betreffenden Teilnehmer möglich, der im Besitz des geheimen Schlüssels k ist, in dieser Weise seine Unterschrift zu leisten.

ElGamal-Signatur: Erklärung

- Es ist einfacher, die ElGamal-Signatur zu verstehen, wenn man mit der Verifikationsgleichung anfängt und daraus die Unterschrift konstruiert.
- Wir starten mit der Kongruenz $a^m \equiv (\tilde{k})^{\tilde{z}} \cdot (\tilde{z})^x \pmod{n}$.
- Mit $\tilde{k} \equiv a^k \pmod{n}$ und $\tilde{z} \equiv a^z \pmod{n}$ erhalten wir

$$a^m \equiv (a^k)^{\tilde{z}} \cdot (a^z)^x \equiv a^{k \cdot \tilde{z} + z \cdot x} \pmod{n}$$

- Da a eine Primitivwurzel modulo n ist, gilt die vorherige Kongruenz gdw.

$$m \equiv k \cdot \tilde{z} + z \cdot x \pmod{(n-1)}$$

- Wenn man diese Kongruenz nach x auflöst, so erhält man die Unterschrift $(\tilde{z}, x)$.

ElGamal-Signatur: Beispiel

- Die Primzahl sei $n = 23$, die Primitivwurzel sei $a = 5$.
- Der Teilnehmer, z.B. Bob, der ein Dokument, z.B. $m = 10$ unterschreiben soll, hat initial das Schlüsselpaar $(k, \tilde{k}) = (15, 19)$ gewählt.
- Zunächst wählt er die Zufallszahl $z = 5$ (teilerfremd zu 22) und berechnet $\tilde{z} = 20$.
- Ferner wird das multiplikative Inverse von z, modulo $n - 1 = 22$, benötigt. Er berechnet mittels extggT: $z^{-1} = 9$.
- Einsetzen in die Formel $x = ((m - k \cdot \tilde{z}) \cdot (z^{-1}) \bmod (n - 1))$ ergibt $x = 8$.
- Die Unterschrift ergibt sich also als $(\tilde{z}, x) = (20, 8)$.
- Dokument plus Unterschrift ist damit $(10, (20, 8))$.

ElGamal-Signatur: Beispiel

- Dokument plus Unterschrift ist $(m, (\tilde{z}, x)) = (10, (20, 8))$.
- Die Primzahl ist $n = 23$ und die Primitivwurzel ist $a = 5$.
- Der öffentliche Schlüssel ist $\tilde{k} = 19$.
- Zum Verifizieren der Unterschrift müssen zwei Terme berechnet und auf Gleichheit überprüft werden.
- Erster Term: $(a^m \bmod n) = (5^{10} \bmod 23) = 9$.
- Zweiter Term: $((\tilde{k})^{\tilde{z}} \cdot (\tilde{z})^x \bmod n) = (19^{20} \cdot 20^8 \bmod 23) = 9$.