Kryptographie

Vorlesungsmanuskript

WS 2017/18

Markus Kirschmer

Lehrstuhl B für Mathematik

RWTH Aachen University

Vorwort

Dieses Manuskript entstand im Rahmen meiner Vorlesung „Kryptographie“ im Wintersemester 2017/18 an der RWTH Aachen. Kenntnisse der Algebra und Zahlentheorie werden nur in sehr beschränktem Umfang vorausgesetzt, wie sie zum Beispiel in der Vorlesung „Computeralgebra“ vermittelt werden. Alle weiteren benötigten Hilfmittel werden vollständig entwickelt. Das gilt insbesondere für die zahlentheoretischen Grundlagen.

Das Ziel dieser Vorlesung ist es, einen Einblick in die Public-Key Kryptographie zu geben, welche auf elementarer Zahlentheorie und elliptischen Kurven basiert. In der Vorlesung geht es nicht darum, diverse praktische Verfahren detailliert zu erklären, vielmehr wird die Fähigkeit vermittelt, sich leicht in spezielle Verfahren einarbeiten zu können.

Das Manuskript basiert in weiten Teilen fast wörtlich auf dem gleichnamigen Skript meines geschätzen Lehrers Herrn Prof. Dr. W. Lütkebohmert [Lüt02]. Die Verantwortung liegt jedoch bei mir allein. Für Hinweise auf Fehler und Unklarheiten im Text bin ich jederzeit dankbar.

Aachen, im September 2017
Markus Kirschmer

Inhaltsverzeichnis

1. Einführung

Bevor wir mit dem eigentlichen Thema beginnen, wollen wir die Begriffe Kryptographie und Codierungstheorie gegeneinander abgrenzen.

In der Kryptographie beschäftigt man sich mit der Verschlüsselung von Daten zwecks Geheimhaltung und Schutz vor unerlaubtem Zugriff; das zentrale Vorhaben ist hier, *Vertraulichkeit* der Daten zu gewährleisten.

In der Codierungstheorie geht es um die sichere Datenübermittlung, also darum, daß korrekt übermittelt werden; es ist zunächst kein Schutz vor fremden Einblick in die Daten vorgesehen.

1.1. Historische Bemerkungen

Wir wollen zunächst einige historische Verschlüsselungsverfahren erwähnen. All diesen Verfahren ist gemein, daß sie *symmetrisch* sind, d.h. bei der Verschlüsselung und der Entschlüsselung kommen dieselben Schlüssel zum Einsatz.

1.1.1. Cäsar-Chiffre

Die Idee, mathematische Methoden zur *Chiffrierung*, d.h. der i*Verschlüsselung* von Texten zu benutzen ist schon bei den alten Römern zu beobachten. Identifiziert man unser Alphabet $\mathbb{A} := \{A, B, \ldots, Z\}$ vermöge

$$f\colon \mathbb{A} \to \mathbb{Z}/26\mathbb{Z},\ A \mapsto 0 + 26\mathbb{Z},\ B \mapsto 1 + 26\mathbb{Z},\ \ldots\ ,\ Z \mapsto 25 + 26\mathbb{Z}$$

mit dem Ring $\mathbb{Z}/26\mathbb{Z}$, so kann man arithmetische Operationen in diesem Ring benutzen, um Wörter in rekonstruierbarer Weise zu verändern. Bei der *Cäsar-Chiffre* zum Beispiel verwendet man die bijektive affine Selbstabbildung

$$e_{a,b}\colon \mathbb{Z}/26\mathbb{Z} \to \mathbb{Z}/26\mathbb{Z},\ x \mapsto a \cdot x + b$$

wobei $a \in (\mathbb{Z}/26\mathbb{Z})^*$ und $b \in \mathbb{Z}/26\mathbb{Z}$.

Beispiel 1.1.1 Angenommen, wir wollen den folgenden *Klartext*[1] verschlüsseln:

```
ACHWA SMUSS MANOF TVONB OESEN KINDE RNHOE RENOD ERLES ENWIE
ZUMBE ISPIE LVOND IESEN WELCH EMAXU NDMOR ITZHI ESSEN
```

Wir wählen dazu z.B. die Abbildung $e_{a,b}$ von oben mit $a = 3 + 26\mathbb{Z}$ und $b = 2 + 26\mathbb{Z}$. Als verschlüsselten Text, auch *Geheimtext*, *Schlüsseltext* oder *Chiffrat* genannt, erhalten wir

```
CIXQC EMKEE MCPSR HNSPF SOEOP GAPLO BPXSO BOPSL OBJOE OPQAO
ZKMFO AEVAO JNSPL AOEOP QOJIX OMCTK PLMSB AHZXA OEEOP
```

Bei der Cäsar-Chiffre handelt es sich um eine *monoalphabetische* Verschlüsselung, d.h. jedes Vorkommen eines Buchstabens im Klartext wird auf denselben Buchstaben im Geheimtext abgebildet. Im obigen Beispiel wird der Buchstabe „A“ immer zu einem „C“, jedes „B“ wird zu einem „F“ und so weiter.

Solche Chiffrierungen sind mittels Häufigkeitsanalysen sehr leicht zu knacken. Man macht sich zunutze, daß im Deutschen wie im Englischen der Buchstabe „E“ mit Abstand am häufigsten ist. Tabelle 1.1 zeigt die Häufigkeitsverteilung der einzelnen Buchstaben im Deutschen, vgl. [Beu05, S. 10].

[1] Die erste Strophe von Wilhelm Buschs Kinderbuch „Max und Moritz“.

Tabelle 1.1.: Häufigkeitsverteilung der einzelnen Buchstaben in der deutschen Sprache

Buchstabe	Häufigkeit	Buchstabe	Häufigkeit	Buchstabe	Häufigkeit
E	17,40 %	U	4,35 %	K	1,21 %
N	9,78 %	L	3,44 %	Z	1,13 %
I	7,55 %	C	3,06 %	P	0,79 %
S	7,27 %	G	3,01 %	V	0,67 %
R	7,00 %	M	2,53 %	ß	0,31 %
A	6,51 %	O	2,51 %	J	0,27 %
T	6,15 %	B	1,89 %	Y	0,04 %
D	5,08 %	W	1,89 %	X	0,03 %
H	4,76 %	F	1,66 %	Q	0,02 %

Im Chiffrat aus Beispiel 1.1.1 ist „O“ der häufigste Buchstabe, gefolgt von „P“. Wir vermuten daher, daß „E“ auf „O“ und „N“ auf „P“ abgebildet wurden. Das liefert folgendes lineares Gleichungssystem über $\mathbb{Z}/26\mathbb{Z}$:

$$a \cdot 4 + b \equiv 14 \pmod{26}$$
$$a \cdot 13 + b \equiv 15 \pmod{26}$$

Damit erhält man sofort die verwendeten Parameter $a = 3 + 26\mathbb{Z}$ und $b = 2 + 26\mathbb{Z}$.

Selbst wenn man keine affine, sondern eine beliebige Bijektion auf $\mathbb{Z}/26\mathbb{Z}$ benutzt, kann der Klartext sehr oft aus dem Chiffrat mittels einer Häufigkeitsanalyse ermittelt werden. In der Praxis verwendet man nicht nur die Verteilung der einzelnen Buchstaben, sondern benutzt auch die Verteilung von Buchstabenpaaren und -tripeln. Die häufigsten Buchstabenpaare im Deutschen sind z.B. „ER“ und „EN“. Ferner folgt auf ein „C“ fast immer ein „H“ oder „K“ usw.

1.1.2. Vignère-Chiffre

Im 16. Jahrhundert hat der Franzose Vignère eine Abwandlung dieses Verfahrens in Form einer Blockchiffrierung angewandt. Bei einer Blockchiffre wird der Klartext in Blöcke fester Länge eingeteilt und diese werden dann getrennt voneinander verschlüsselt. Konkret wählte er ein *Codewort* der Länge n, mit obiger Identifikation also einen Vektor $b \in (\mathbb{Z}/26\mathbb{Z})^n$. Der Klartext wird nun in Blöcke der Länge n eingeteilt. Jeder Block entspricht also einem Vektor $x \in (\mathbb{Z}/26\mathbb{Z})^n$. Vignères Verschlüsselung ist dann die Translation

$$(\mathbb{Z}/26\mathbb{Z})^n \to (\mathbb{Z}/26\mathbb{Z})^n,\ x \mapsto x + b\,.$$

Beispiel 1.1.2 Wir wollen den Klartext aus Beispiel 1.1.1 mit den Schlüssel „BOECK“[2] verschlüsseln:

```
KLARTEXT:  ACHWA SMUSS MANOF TVONB OESEN KINDE RNHOE RENOD ERLES ENWIE
SCHLÜSSEL: BOECK BOECK BOECK BOECK BOECK BOECK BOECK BOECK BOECK BOECK
CHIFFRAT:  BQLYK TAYUC NORQP UJSPL PSWGX LWRFO SBLQO SSRQN FFPGC FBAKO

           ZUMBE ISPIE LVOND IESEN WELCH EMAXU NDMOR ITZHI ESSEN
           BOECK BOECK BOECK BOECK BOECK BOECK BOECK BOECK BOECK
           AIQDO JGTKO MJSPN JSWGX XSPER FAEZE ORQQB JHDJS FGWGX
```

Die Vignère-Chiffre ist nicht monoalphabetisch, man spricht von einer *polyalphabetischen* Verschlüsselung. Dadurch erreicht sie eine wesentlich gleichmäßigere Verteilung der Buchstaben im Chiffrat. Mit einer Frequenzanalyse kann man diese Verschlüsselung nicht direkt knacken. Man kann die Vignère-Chiffre aber als Kombination von n Cäsar-Chiffren auffassen. Dazu sei $1 \le k \le n$ und es sei $b = (b_1, \cdots, b_n)$. Die Buchstaben des Klartextes, deren Indizes in der arithmetischen Progression $k + n\mathbb{Z} = \{k + \ell n\,;\, \ell \in \mathbb{Z}\}$ liegen, werden nämlich alle mit der Abbildung e_{1,b_k} aus Beispiel 1.1.1 verschlüsselt. Man sieht also, daß man zum Knacken einer Vignère-Chiffre im wesentlichen nur die verwendete Blocklänge n richtig raten muß. Danach verfährt man wie bei der Cäsar-Chiffre.

[2] Der Name des Schneidermeisters in Max und Moritz drittem Streich.

Wir wollen das an Beispiel 1.1.2 einmal versuchen. Im Chiffrat finden wir den Text „WGX“ drei Mal, nämlich an den Positionen 23, 68 und 93. Dies kann von drei gleichen Stellen in Klartext herrühren, die auf dieselbe Weise verschlüsselt wurden. Das heißt, die Differenzen der Fundstellen sind Vielfache von n. Also ist n ein Teiler von 45, 25 und 70. Da wir $n = 1$ ausschließen, bleibt noch $n = 5$. Das Dechiffrieren der 5 individuellen Cäsar-Chiffren sei dem geneigten Leser zur Übung überlassen.

1.1.3. Hill-Chiffre

Hill stellt 1929 in [Hil29] fest, daß Transformationen mit einer invertierbaren Matrix $A \in \mathrm{GL}_n(\mathbb{Z}/26\mathbb{Z})$, also $x \mapsto Ax + b$ wesentlich sicherer als die einfachen Transformationen von Vignère sind. Wenn man die inverse Matrix A^{-1} kennt, so rekonstruiert man den Klartext auch sehr leicht. Eine Attacke auf diese Chiffrierung ist nicht ganz so leicht, da man sie nicht durch eine Frequenzanalyse bekommt. Da das Verfahren jedoch als lineares Gleichungssystem beschrieben ist, kann man das Verfahren leicht knacken, wenn man etwas Klartext und den zugehörigen verschlüsselten Text kennt.

1.2. AES

Der *Advanced Encryption Standard* (AES) wurde Ende 2000 vom US National Institute of Standards and Technology (NIST) als Nachfolger des in die Tage gekommenen DES-Verfahrens bekannt gegeben. Es ist eine spezielle Version des Rijndael-Algorithmus, benannt nach seinen Entwicklern Joan Daemen und Vincent Rijmen. AES ist das zur Zeit meist genutzte symmetrische Verschlüsselungsverfahren. Eine detaillierte Beschreibung findet sich in der Spezifikation des Standards [AES].

Bei AES handelt es sich ebenfalls um eine symmetrische Blockchiffre. Die Blocklänge ist bei AES stets 128 Bits, also 16 Bytes. Die verwendeten Schlüssel haben eine Länge von 128, 192 oder 256 Bits.

Für die mathematische Beschreibung des AES-Verfahrens benötigen wir etwas Notation. Das Polynom $f := X^8 + X^4 + X^3 + X + 1 \in \mathbb{F}_2[X]$ ist irreduzibel. Das verifiziert man zum Beispiel direkt mit den Faktorisierungsmethoden aus der Vorlesung „Computeralgebra“ oder aber man teilt f mittels Polynomdivision durch die Polynome in $\mathbb{F}_2[X]$ vom Grad ≤ 4. Somit ist

$$K := \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$$

ein Körper mit 2^8 Elementen.

Ein Byte, also 8 Bits $(b_7, \ldots, b_0)^{\mathrm{t}} \in \mathbb{F}_2^8$ identifizieren wir mit einem Element aus K via

$$\mathbb{F}_2^8 \to K,\ (b_7, \ldots, b_0)^{\mathrm{t}} \mapsto \sum_{i=0}^{7} b_i \cdot \overline{X}^i\,. \tag{1.2.1}$$

Ein Tupel von 16 Bytes $(b_0, \ldots, b_{15})$ identifizieren wir mit einer 4×4-Matrix über K vermöge

$$K^{16} \to K^{4\times 4},\ (b_0, \ldots, b_{15})^{\mathrm{t}} \mapsto \begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{pmatrix} \tag{1.2.2}$$

Das AES-Verfahren besteht aus den folgenden vier bijektiven Selbstabbildungen auf $K^{4\times 4}$.

1. Die Funktion $\texttt{AddRoundKeys}_C$ ist die Addition mit $C \in K^{4\times 4}$, also
$$\texttt{AddRoundKeys}_C\colon K^{4\times 4} \to K^{4\times 4},\ B \mapsto B + C\,.$$
2. Die Funktion `ShiftRows` verschiebt die i-te Zeile einer Matrix um $(i-1)$ Einträge nach links, also
$$\texttt{ShiftRows}\colon K^{4\times 4} \to K^{4\times 4},\ \begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{pmatrix} \mapsto \begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_5 & b_9 & b_{13} & b_1 \\ b_{10} & b_{14} & b_2 & b_6 \\ b_{15} & b_3 & b_7 & b_{11} \end{pmatrix}$$

3. Die Funktion
$$\texttt{MixColumns}\colon K^{4\times 4} \to K^{4\times 4},\ B \mapsto AB$$
ist die Linksmultiplikation mit der invertierbaren Matrix
$$A := \begin{pmatrix} \overline{X} & \overline{X}+1 & 1 & 1 \\ 1 & \overline{X} & \overline{X}+1 & 1 \\ 1 & 1 & \overline{X} & \overline{X}+1 \\ \overline{X}+1 & 1 & 1 & \overline{X} \end{pmatrix} \in \mathrm{GL}_4(K)\,.$$

4. Die Funktion
$$\texttt{SubBytes}\colon K^{4\times 4} \to K^{4\times 4},\ (x_{i,j})_{i,j} \mapsto (S(x_{i,j}))_{i,j}$$
ist das elementweise Anwenden der *Substitutionsbox*
$$S\colon K \to K,\ a \mapsto \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix} \cdot a^{-1} + \begin{pmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{pmatrix}$$
wobei man K wie in (1.2.1) mit $\mathbb{F}_2^8$ identifiziert und 0^{-1} als 0 interpretiert. Man kann durch Ausprobieren aller Werte leicht verifizieren, daß die Funktion S invertierbar und fixpunktfrei ist.

Das AES-Verfahren ist mit obiger Notation nun schnell erklärt. Es folgt folgendem Schema.

Algorithmus 1.2.1: AES
Eingabe: 16 Bytes $(b_0, \ldots, b_{15})$ sowie ein Schlüssel k der Länge $\ell = 128$, 192 oder 256 Bit.
Ausgabe: Die mit k verschlüsselten 16 Bytes.

Setze $R = 10$, falls $\ell = 128$; $R = 12$ falls $\ell = 192$ und $R = 14$ falls $\ell = 256$.
$k_0, \ldots, k_R \leftarrow \texttt{KeyExpansion}(k)$ `// generiert` $(R+1)$ `Teilschlüssel der Länge 128 Bit`
Fasse $(b_0, \ldots, b_{15})$ sowie die Teilschlüssel $k_0, \ldots, k_R$ als Matrizen $B, C_0, \ldots, C_R$ in $K^{4\times 4}$ auf, vgl. Gl. (1.2.2).
$B \leftarrow \texttt{AddRoundKeys}_{C_0}(B)$
for $1 \le i \le R$ **do**
    $B \leftarrow \texttt{SubBytes}(B)$
    $B \leftarrow \texttt{ShiftRows}(B)$
    if $i < R$ **then** $B \leftarrow \texttt{MixColumns}(B)$
    $B \leftarrow \texttt{AddRoundKeys}_{C_i}(B)$
return die Einträge von B aufgefaßt als 16 Bytes, vgl. Gleichung (1.2.2).

Für die Beschreibung der Funktion `KeyExpansion` sei auf die Spezifikation des Standards [AES] verwiesen. Bei der Entschlüsselung wendet man die Inversen der bijektiven Abbildungen `AddRoundKeys`, `SubBytes`, `ShiftRows` und `MixColumns` in umgekehrter Reihenfolge an.

Zu den Vorteilen des AES-Verfahrens zählt unter anderem, daß es sehr leicht zu implementieren ist, die Referenzimplementierung in C benötigt z.B. nur 500 Zeilen. Das liegt zum einen an seinem sehr einfachen Aufbau, zum anderen an der Tatsache, daß der Körper K nicht explizit über Polynome oder Ähnliches realisieren werden muß. Dazu geht man wie folgt vor:

1. Die Substitutionsbox S kann vorausberechnet und als 256 Byte große Lookup-Table abgespeichert werden.
2. Die Addition in K kann als XOR der entsprechenden Bitmuster realisiert werden.
3. Die Funktion `MixColumns` benötigt nur die Multiplikation mit $\overline{X}$. Das kann leicht durch Verschieben der Bitmustern erreicht werden.

4. Beim Entschlüsseln braucht man auch die Multiplikation mit anderen Elementen von K. Das kann leicht mittels des Distributivgesetzes auf Additionen und Multiplikationen mit $\overline{X}$ reduziert werden.

Der Algorithmus benötigt also sehr wenig Speicher und Rechenleistung. Insbesondere läßt er sich auch auf Chipkarten etc. einsetzen. Ferner verfügen praktisch alle modernen handelsüblichen Prozessoren für PCs spezielle Befehle, die die Ver-/Entschlüsselung mittels AES beschleunigen. Sie verarbeiten typischerweise mehrere hundert Megabyte pro Sekunde.

Gerade der sehr einfache Aufbau von AES liefert aber auch Anlaß zur Kritik:

1. Lediglich die Funktion `AddRoundKeys` hängt vom gewählten Schlüssel ab.
2. Die Funktion `SubBytes` ist die einzige nichtlineare Funktion.
3. Während `SubBytes` zwar verhindert, daß das AES-Verfahren als lineares Gleichungssystem beschrieben werden kann, so kann es doch als (überbestimmtes) quadratisches Gleichungssystem beschrieben werden.

Zur Zeit ist jedoch kein in der Praxis relevanter Angriff auf AES bekannt.

1.3. Public-Key Kryptographie

Bis 1979 waren alle Krypto-Systeme symmetrisch, d.h. Sender und Empfänger hatten sich auf ein System geeignet und die Schlüssel waren beiden bekannt. Man spricht hier auch von *private keys*. Jeder, der hinreichend viel Information über das verwendete System hat, kann die verschlüsselten Nachrichten auch entschlüsseln. Das Problem ist also, die verwendeten Schlüssel sicher auszutauschen und vor anderen geheim zu halten.

Das ändert sich 1976 dramatisch mit der Idee von Diffie und Hellman, vgl. [DH76]. Zu Grunde liegt die Idee, eine Einwegfunktion zur Chiffrierung zu benutzen.

Definition 1.3.1 Eine injektive Abbildung $f\colon X \to Y$ heißt *Einwegfunktion*, wenn $f(x)$ für $x \in X$ leicht zu berechnen ist, aber das Urbild $f^{-1}(y)$ für $y \in Y$ nur sehr langwierig zu bestimmen ist, solange $y \in Y$ zufällig gewählt ist.

Das ist zwar keine übliche mathematische Funktion; dafür müßte man etwas über Komplexität erläutern, aber es ist vielleicht doch klar, was gemeint ist.

In der Kryptographie benutzt man eine spezielle Klasse von Einwegfunktionen, die nur dann Einwegfunktionen sind, wenn man gewisse Zusatzinformationen geheim hält. Falls man diese Zusatzinformationen hat, so sollen Urbilder leicht zu berechnen sein.

Der Vorteil einer solcher Chiffrierung liegt auf der Hand. Man kann die Funktion öffentlich machen. Somit kann jeder dem Besitzer der Zusatzinformation Daten verschlüsselt senden, ohne daß vorher eine Absprache getroffen werden muß. Dazu nun das entscheidende Beispiel, das die Fachwelt überzeugt hat.

Beispiel 1.3.2 (RSA-Verfahren) Es sei

$$N = p \cdot q$$

ein Produkt zweier großer Primzahlen mit etwa 150 Dezimalstellen. Dann gilt

$$\varphi(N) = (p-1)\cdot(q-1) = \#\left(\mathbb{Z}/N\mathbb{Z}\right)^* .$$

Weiterhin wählen wir eine Zahl $1 < e < \varphi(N)$ mit

$$\mathrm{ggT}(e, \varphi(N)) = 1 .$$

Es sollte e auch hinreichend groß sein, also z.B. $e > 10^{10}$. Dann existiert eine natürliche Zahl d mit

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

Die Zahlen N und e veröffentlicht man; sie bilden den öffentlichen Schlüssel (*public key*). Die Zahl d aber hält man geheim.

Das RSA-Verfahren besteht nun im Wesentlichen aus den Abbildungen

$$(Encryption) \qquad E\colon \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z},\ x \mapsto x^e$$
$$(Decryption) \qquad D\colon \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z},\ x \mapsto x^d$$

Man rechnet nämlich leicht nach, daß

$$D \circ E = \mathrm{id}_{\mathbb{Z}/N\mathbb{Z}} = E \circ D\,.$$

Wir werden noch sehen, daß man das Potenzieren in $\mathbb{Z}/N\mathbb{Z}$ sehr schnell, d.h. in maximal $2\log_2(N)$ Rechenoperationen durchführen kann, vgl. **??**. Mittels der öffentlichen Schlüssel N und e kann einem jeder nun eine verschlüsselte Nachricht senden.
Die Kenntnis von d ist äquivalent zur Kenntnis von p und q. Letztere kann man man nach heutigem Stand der Theorie und Technik nur sehr schwer aus N und e bestimmen. Daher ist es fast unmöglich, die Funktion E ohne Kenntnis von d umzukehren.
Beim RSA-Verfahren kommen zur Ver- und Entschlüsselung also verschiedene Schlüssel zum Einsatz. Man spricht daher von einem *asymmetrischen* Verfahren. Für weitere Details zum RSA-Verfahren sei auf **??** verwiesen.

Man kann sich natürlich fragen, warum es so lange gedauert hat, bis man auf diese Methode gestoßen ist, wenn doch nur mathematische Sachverhalte benutzt werden, die Leonhard Euler schon vor ca. 200 Jahren bestens bekannt waren. Ein möglicher Grund ist vielleicht, daß Kryptographie nur im militärischen und diplomatischen Bereich benutzt wurde und man dort mit den symmetrischen Verfahren ausreichend gut bedient war. Ein anderer ist vielleicht die Größe der Zahlen mit denen man rechnen muß. Solche Rechnungen lassen sich nicht von Hand durchführen, dazu benötigt man Computer. Ende der 70er war ein Bedarf an Kryptographie im kommerziellen elektronischen Handel aufgekommen und es waren Computer verfügbar, die die entsprechenden Rechnungen schnell durchführen konnten.

Es sei noch angemerkt, daß symmetrische Verfahren meist sehr viel schneller sind als Public-Key Verfahren. In der Praxis verwendet man daher einen Hybridansatz um große Datenmengen zu verschlüsseln. Man verschlüsselt die Daten zunächst mit einem schnellen symmetrischen Verfahren wie zum Beispiel AES. Der dabei verwendete Schlüssel (welcher ja nur wenige Bytes groß ist), wird dann mit einem Public-Key Verfahren wie zum Beispiel RSA verschlüsselt.

1.4. Einsatz von Public-Key Kryptographie

Man kann Public-Key Kryptographie anwenden für:

1. Vertrauliche Datenübertragung
2. Authentifikation von Nachrichten; Elektronische Unterschrift
3. Schlüsselaustausch
4. Geheimnisteilung
   Zum Beispiel gibt es Entscheidungen, die von k Personen getroffen werden dürfen aber nicht schon von $(k-1)$ Personen. Die Personen müssen dabei nicht alle an einem Ort sein.
5. Zero Knowledge Proof
   Hier geht es darum, jemanden davon zu überzeugen, daß man eine Information hat, ohne dem anderen die Information mitzuteilen.

Diese Aufgaben werden durch verschiedene Protokolle verwirklicht. Ein *Protokoll* ist lediglich ein festgelegtes Prozedere, wie die Daten gegenseitig ausgetauscht werden.

In dieser Vorlesung geht es nicht darum, diverse praktische Verfahren detailliert zu erklären, vielmehr wird die Fähigkeit vermittelt, sich leicht in spezielle Verfahren einarbeiten zu können. Konkrete Verfahren findet zu man zu Genüge im Buch von Menezes, van Oorschot und Vanstone [MOV01]. Insbesondere wollen wir uns im wesentlichen nur mit Public-Key Verfahren beschäftigen, die auf elementarer Zahlentheorie und elliptischen Kurven basieren.

2. Zahlentheoretische Grundlagen

2.1. Teilbarkeit im Ring der ganzen Zahlen

Definition 2.1.1 Es seien $a, b \in \mathbb{Z}$ ganze Zahlen.

1. Die ganze Zahl b heißt *Teiler* von a, wenn es eine ganze Zahl $c \in \mathbb{Z}$ gibt mit $a = c \cdot b$. In dem Fall schreibt man $b \mid a$. Ist b kein Teiler von a, so schreibt man $b \nmid a$.
2. Eine ganze Zahl $p \geq 2$ heißt *Primzahl*, wenn p nur die beiden trivialen positiven Teiler 1 und p hat.
3. Eine ganze Zahl $g \in \mathbb{Z}$ heißt *gemeinsamer Teiler* von a und b, falls g sowohl a als auch b teilt.
4. Eine natürliche Zahl $g \in \mathbb{N}$ heißt *größter gemeinsamer Teiler* von a und b, falls
   a) g ein gemeinsamer Teiler von a und b ist.
   b) Für jeden gemeinsamen Teiler d von a und b gilt $d \mid g$.

   Wir schreiben $\mathrm{ggT}(a, b)$ für den größten gemeinsamen Teiler von a und b. Man sagt, a und b sind *teilerfremd*, falls $\mathrm{ggT}(a,\, b) = 1$.

In den Grundlagenvorlesungen haben Sie bereits gesehen, daß man größte gemeinsame Teiler durch Division mit Rest bestimmen kann. Hat man zwei ganze Zahlen $a, b \in \mathbb{Z}$ der Bitlänge n, so benötigt eine naive Implementierung dieser Methode $O(n^3)$ Bitoperationen. Die folgende Variante macht es etwas besser.

Algorithmus 2.1.2: Erweiterter größter gemeinsamer Teiler
Eingabe: Ganze Zahlen $a, b \in \mathbb{Z}$.
Ausgabe: Ein Tripel $(g, \lambda, \mu) \in \mathbb{Z}^3$ mit $g = \mathrm{ggT}(a, b)$ und $g = \lambda \cdot a + \mu \cdot b$.

if $a < 0$ *oder* $b < 0$ **then return** $(g,\, \mathrm{sgn}(a) \cdot \lambda,\, \mathrm{sgn}(b) \cdot \mu)$ wobei $(g, \lambda, \mu) =$ `XGCD(`$|a|$, $|b|$`)`
if $a = 0$ **then return** $(b, 0, 1)$
if $b = 0$ **then return** $(a, 1, 0)$
if a *und* b *beide gerade* **then return** $(2g, \lambda, \mu)$ wobei $(g, \lambda, \mu) =$ `XGCD(`$a/2$, $b/2$`)`
else if a *gerade* **then**
    $(g, \lambda, \mu) \leftarrow$ `XGCD(`$a/2$, b`)`
    if λ *ungerade* **then** $\lambda \leftarrow \lambda + b$, $\mu \leftarrow \mu - a/2$
    return $(g, \lambda/2, \mu)$.
else if b *gerade* **then**
    $(g, \lambda, \mu) \leftarrow$ `XGCD(`a, $b/2$`)`
    if μ *ungerade* **then** $\lambda \leftarrow \lambda - b/2$, $\mu \leftarrow \mu + a$
    return $(g, \lambda, \mu/2)$.
else if $a < b$ **then return** $(g,\, \lambda - \mu,\, \mu)$ wobei $(g,\, \lambda,\, \mu) =$ `XGCD(`a, $b - a$`)`
else return $(g,\, \lambda,\, \mu - \lambda)$ wobei $(g,\, \lambda,\, \mu) =$ `XGCD(`$a - b$, b`)`

Beweis. Wir müssen zeigen, daß der Algorithmus terminiert und das richtige Ergebnis liefert. Ohne Einschränkung gilt $a, b \geq 0$. Da $a + b$ mit jeder Rekursion kleiner wird, terminiert der Algorithmus. Ist $a = 0$ oder $b = 0$, so liefert er sicher das richtige Ergebnis. Die anderen Fälle reduziert man mittels **??** darauf. Sind $a, b > 0$ beide gerade, so ist $\mathrm{ggT}(a/2, b/2) = 2\,\mathrm{ggT}(a, b)$ und $\mathrm{ggT}(a/2, b/2) = \lambda \cdot a/2 + \mu \cdot b/2$ impliziert $\mathrm{ggT}(a, b) = \lambda \cdot a + \mu \cdot b$. Ist a gerade und b ungerade, so gilt $\mathrm{ggT}(a, b) = \mathrm{ggT}(a/2, b)$. Insbesondere folgt aus $\mathrm{ggT}(a/2, b) = \lambda \cdot a/2 + \mu \cdot b$ daher $\mathrm{ggT}(a, b) = \lambda/2 \cdot a + \mu \cdot b = (\lambda + b)/2 \cdot a + (\mu - a) \cdot b$. Die anderen Fälle gehen analog. □

Wir halten weitere direkte Folgerungen aus Algorithmus 2.1.2 fest.

Notiz 2.1.3 Es seien $a, b \in \mathbb{Z}$ ganze Zahlen von Bitlänge n. Die Bestimmung von $\mathrm{ggT}(a, b)$ mit Algorithmus 2.1.2 benötigt $O(n^2)$ Bitoperationen.

Beweis. Ohne Einschränkung sind a und b positiv. Ist a gerade, so wird a halbiert. Das selbe gilt für b. Im Fall, daß a und b beide ungerade sind, ersetzt man a oder b durch $|a-b|$. In der nächsten Rekursion wird dieser Wert dann auf jeden Fall durch 2 geteilt. Damit benötigen der Algorithmus maximal $O(n)$ Rekursionen. Bei jeder Rekursion wird eine feste Anzahl von Vergleichen, Additionen, Subtraktionen sowie Multiplikationen bzw. Divisionen mit 2 durchgeführt. Jede dieser Rechenschritte benötigt $O(n)$ Bitoperationen. □

Korollar 2.1.4 (Bézouts Identität) *Sind $a, b \in \mathbb{Z}$ ganze Zahlen, so gibt es ganze Zahlen $\lambda, \mu \in \mathbb{Z}$ mit*

$$\mathrm{ggT}(a,\, b) = \lambda \cdot a + \mu \cdot b\,.$$

Das folgende Korollar erklärt den Namen „größter gemeinsamer Teiler“ und zeigt seine Eindeutigkeit.

Korollar 2.1.5 *Es seien $a, b \in \mathbb{Z}$ ganze Zahlen. Der größte gemeinsamer Teiler von a und b ist die eindeutig bestimmte natürliche Zahl $g \in \mathbb{N}$ mit*

$$g\mathbb{Z} = a\mathbb{Z} + b\mathbb{Z}\,.$$

Ferner gilt

$$\mathrm{ggT}(a,b) = \begin{cases} 0 & \text{falls } a = b = 0, \\ \max\{d \in \mathbb{Z}\,;\, d \mid a \text{ und } d \mid b\} & \text{sonst.} \end{cases}$$

Beweis. Jeder gemeinsame Teiler d von a und b erfüllt $a, b \in d\mathbb{Z}$. Insbesondere gilt $a\mathbb{Z} + b\mathbb{Z} \subseteq \mathrm{ggT}(a,b)\mathbb{Z}$. Bézouts Identität zeigt die umgekehrte Inklusion.
Nun zur zweiten Identität. Im Fall $a = b = 0$ ist nichts zu zeigen, denn $\mathrm{ggT}(0,0) = 0$. Es sei nun $a \neq 0$ oder $b \neq 0$. Weiter sei $h := \max\{d \in \mathbb{Z}\,;\, d \mid a \text{ und } d \mid b\}$. Es gilt $\mathrm{ggT}(a,b) \leq h$ nach Wahl von h. Umgekehrt teilt h nach Definition $\mathrm{ggT}(a,b)$. Also gilt $h \leq \mathrm{ggT}(a,b)$. Zusammen folgt $h = \mathrm{ggT}(a,b)$. □

Satz 2.1.6 (Euklid) *Teilt eine Primzahl p ein Produkt $a \cdot b$ zweier ganzer Zahlen $a, b \in \mathbb{Z}$, so teilt p einen der beiden Faktoren.*

Beweis. Ohne Einschränkung ist p kein Teiler von a, d.h. $\mathrm{ggT}(a,p) = 1$. Dann gibt es $\lambda, \mu \in \mathbb{Z}$ mit $1 = a\lambda + \mu p$. Damit teilt p die Zahl $\lambda \cdot ab + \mu p \cdot b = (\lambda a + \mu p) \cdot b = b$. □

Satz 2.1.7 (Hauptsatz der elementaren Zahlentheorie) *Jede ganze Zahl $n \in \mathbb{Z} - \{0\}$ besitzt eine eindeutige Produktdarstellung*

$$n = \varepsilon \cdot p_1^{e_1} \cdot \ldots \cdot p_r^{e_r}$$

mit Primzahlen $p_1 < p_2 < \ldots < p_r$, positiven ganzen Zahlen $e_1, \ldots, e_r$ und $\varepsilon \in \{\pm 1\}$. Man nennt diese Darstellung die Primfaktorzerlegung von n.

Beweis. Die Existenz einer solchen Darstellung folgt per Induktion über $|n|$. Ist $|n|$ gleich 1 oder aber eine Primzahl, so ist nichts zu zeigen. Andernfalls ist $n = a \cdot b$ mit $1 < |a| < |n|$ und $1 < |b| < |n|$. Nach Induktionsvoraussetzung besitzen a und b eine Primfaktorzerlegung. Damit gilt dies auch für $n = a \cdot b$.
Zur Eindeutigkeit betrachten wir eine weitere Zerlegung

$$n = \varepsilon' \cdot q_1^{f_1} \cdot \ldots \cdot q_s^{f_s}$$

mit Primzahlpotenzen $q_1^{f_1}, \ldots, q_s^{f_s}$ und $\varepsilon' \in \{\pm 1\}$. Nach Satz 2.1.6 teilt p_1 eine der Primzahlen q_i. Nach Umnumerierung gilt $p_1 \mid q_1$ und somit $p_1 = q_1$. Nun wiederholt man das Argument mit

$$n' = \varepsilon \cdot p_1^{e_1 - 1} \cdot \ldots \cdot p_r^{e_r} = \varepsilon' \cdot q_1^{f_1 - 1} \cdot \ldots \cdot q_s^{f_s}\,.$$

Wegen $|n'| < |n|$ folgt die Behauptung durch Induktion. □

Als Folgerung des Hauptsatz der elementaren Zahlentheorie erhält man sofort den folgenden Satz.

Satz 2.1.8 *Es sei p eine Primzahl. Ist $x \in \mathbb{Q}^*$ eine rationale Zahl, so gibt es genau eine ganze Zahl $\nu_p(x) \in \mathbb{Z}$, so daß $x = p^{\nu_p(x)} \cdot \frac{a}{b}$ mit ganzen Zahlen $a, b \in \mathbb{Z}$ und $p \nmid (a \cdot b)$. Setzt man noch $\nu_p(0) = \infty$, so erhält man eine Abbildung*

$$\nu_p \colon \mathbb{Q} \to \mathbb{Z} \cup \{\infty\} ,$$

die sogenannte p-adische Bewertung. *Für $x, y \in \mathbb{Q}$ gilt:*

1. $\nu_p(x) = \infty \iff x = 0$.
2. $\nu_p(x \cdot y) = \nu_p(x) + \nu_p(y)$.
3. $\nu_p(x + y) \leq \min(\nu_p(x), \nu_p(y))$.
4. *Ist $\nu_p(x) \neq \nu_p(y)$, so gilt $\nu_p(x + y) = \min(\nu_p(x), \nu_p(y))$.*

Beweis. Die Wohldefiniertheit von $\nu_p(x)$ folgt aus dem Hauptsatz der elementaren Zahlentheorie. Nun zu den vier Eigenschaften von ν_p. Die erste und dritte Eigenschaft sind klar und die zweite folgt erneut aus dem Hauptsatz. Wir zeigen die vierte. Diese ist klar, falls $x = 0$ oder $y = 0$. Es sei daher $x \neq 0$ und $y \neq 0$. Nach Reskalieren und der zweiten Eigenschaft dürfen wir $x, y \in \mathbb{Z}$ annehmen. Also ist $x = p^i \cdot a$ und $y = p^j \cdot c$ mit $a, c \in \mathbb{Z}$ und $p \nmid a \cdot c$. Ohne Einschränkung ist $i < j$. Dann ist $x + y = p^i \cdot (a + p^{j-i}c)$. Wegen $\mathrm{ggT}(a + p^{j-i}c, p) = \mathrm{ggT}(a, p) = 1$ ist $\nu_p(x + y) = i = \nu_p(x)$. □

A. Anhang

A.1. Landausche O-Notation

Definition A.1.1 Es sei I eine nach oben unbeschränkte Teilmenge von $\mathbb{R}$. Weiter seien $f, g\colon I \to \mathbb{R}$ zwei reellwertige Funktionen auf I. Man schreibt $f \in O(g)$ oder auch $f = O(g)$, falls

$$\limsup_{x\to\infty} \frac{|f(x)|}{|g(x)|} < \infty\,.$$

In der Praxis entscheidet man $f \in O(g)$ meist nicht mittels der Definition, sondern man benutzt folgende Rechenregeln.

Notiz A.1.2

1. Es seien $f, g\colon I \to \mathbb{R}$ wie in Definition A.1.1. Dann ist $f \in O(g)$ äquvalent dazu, daß es eine Konstante $c \in \mathbb{R}$ und ein $x_0 \in I$ gibt mit
$$|f(x)| \le c \cdot |g(x)| \quad \text{für alle } x \in I \text{ mit } x \ge x_0.$$
2. Ist $c \in \mathbb{R}$, so gilt $c \in O(1)$.
3. Ist $f_1 \in O(g_1)$ und $f_2 \in O(g_2)$, so gilt $f_1 f_2 \in O(g_1 g_2)$.
4. Ist $f \in O(g)$ und $c \in \mathbb{R}$, so ist $cf \in O(g)$.
5. Ist $f_1 \in O(g_1)$ und $f_2 \in O(g_2)$, so gilt $f_1 + f_2 \in O(|g_1| + |g_2|)$.
6. Ist $f \in O(g)$ und $g \in O(h)$, so ist $f \in O(h)$.

Beispiele A.1.3

1. Es sei $f(n) = 3\log(n) + 3n^2 + 2n^3$ für alle positiven ganzen Zahlen n. Wegen $\log(n) \in O(n^3)$ und $n^2 \in O(n^3)$ ist $f \in O(n^3)$.
2. Es sei $b > 0$ mit $b \neq 1$. Für $x > 0$ ist $\log_b(x) = \frac{\log(x)}{\log(b)}$. Daher gilt $f \in O(\log(x)) \iff f \in O(\log_b(x))$.

A.2. Komplexität von Algorithmen für ganze Zahlen

Eine natürliche Zahl N wird in einem Computer als Binardarstellung von $|N|$ und ihrem Vorzeichen abgespeichert. Für die Anzahl $n = n(N)$ der Binärstellen von N gilt daher $n(N) \in O(\log(|N|))$.

Wir wollen die Komplexität von Algorithmen für ganze Zahlen in einzelnen Bitoperationen messen. Das ist zwar nicht ganz korrekt, da moderne Prozessoren mehrere hundert Bits simultan manipulieren können, für unsere Zwecke soll dies jedoch genügen.

Definition A.2.1 Es sei A ein Algorithmus, der als Eingabe ganze Zahlen der Bitlänge n nimmt. Weiter sei $f(n)$ die Anzahl der Bitoperationen die der Algorithmus ausführt. Man sagt A ist *polynomiell*, falls $f \in O(n^k)$ für ein $k \in \mathbb{N}$.

Um die Komplexität solcher Algorithmen abschätzen zu können, muß man wissen, wie schnell man für zwei ganze Zahlen der Bitlänge n diverse Grundoperationen durchführen kann. Bereits die in der Schule (im Fall des Zehnersystems) erlernten Methoden wie Addition mit Übertrag, schriftliche Multiplikation und Division, etc. liefern die in Tabelle A.1 erwähnten Laufzeiten.

Tabelle A.1.: Laufzeiten der Grundrechenarten

Operation	Input	Bitoperationen
Vergleich	Zwei ganze Zahlen der Bitlänge n	$O(n)$
Addition	Zwei ganze Zahlen der Bitlänge n	$O(n)$
Subtraktion	Zwei ganze Zahlen der Bitlänge n	$O(n)$
Multiplikation	Zwei ganze Zahlen der Bitlänge n	$O(n^2)$
Division mit Rest	Zwei ganze Zahlen der Bitlänge n	$O(n^2)$
Multiplikation mit 2^k	Ganze Zahl der Bitlänge n und Exponent $k \in \mathbb{N}$	$O(n+k)$
Division durch 2^k	Ganze Zahl der Bitlänge n und Exponent $k \in \mathbb{N}$	$O(n+k)$

Insbesondere sind diese Grundoperationen alle polynomiell. Die Komplexität der Schulmethoden sind im Fall der Multiplikation und Division mit Rest jedoch nicht optimal. Der Karatsuba-Algorithmus zum Beispiel multipliziert zwei ganze Zahlen der Bitlänge n in $O(n^{\log_2(3)})$ Bitoperationen. Es gibt sogar asympotisch noch bessere Verfahren, deren Geschwindigkeitsvorteil jedoch erst ab mehreren Tausend Binärstellen zum Vorschein tritt. Sie sind daher für diese Vorlesung irrelevant. Man kann auch zeigen, daß die Division mit Rest mit der selben Komplexität durchgeführt werden kann wie Multiplikationen.

Der Einfachheit halber vereinbaren wir für die Vorlesung die Komplexitäten wie in Tabelle A.1.

Literatur

[AES] United States National Institute of Standards and Technology (NIST). *Specification for the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. 2001. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[Beu05] Albrecht Beutelspacher. *Kryptologie*. 7. Aufl. Wiesbaden: Vieweg Verlagsgesellschaft, 2005.

[DH76] Whitfield Diffie und Martin Hellman. „New directions in cryptography". In: *IEEE Transactions on Information Theory* 22 (1976), S. 644–654.

[Hil29] Lester S. Hill. „Cryptography in an Algebraic Alphabet". In: *The American Mathematical Monthly* 36.6 (1929), S. 306.

[Lüt02] Werner Lütkebohmert. „Kryptographie". Vorlesungsmanuskript, Universität Ulm. SS2002.

[MOV01] Alfred J. Menezes, Paul C. van Oorschot und Scott A. Vanstone. *Handbook of Applied Cryptography*. 5. Aufl. CRC press, 2001.

Symbolverzeichnis

Bezeichnung	Beschreibung
$\mathbb{Z}$	Ring der ganzen Zahlen.
$\mathbb{N}$	Halbring der natürlichen Zahlen $\{0, 1, 2, \dots\}$.
$\mathbb{Q}$	Körper der rationalen Zahlen.
$\mathbb{F}_q$	Körper mit q Elementen.
$\mathbb{R}$	Körper der reellen Zahlen.
$\mathbb{C}$	Körper der komplexen Zahlen.
R^*	Einheitengruppe eines Rings R, d.h. $R^* = \{r \in R\,;\, rs = sr = 1 \text{ für ein } s \in R\}$.
$R[X]$	Polynomring über dem kommutativen Ring R mit Unbestimmter X.
$\lvert x\rvert$	Absolutbetrag von x.
$\operatorname{sgn} x$	Vorzeichen von x.
$\lceil x\rceil$	Aufrunden, also $\lceil x\rceil = \min\{y \in \mathbb{Z}\,;\, y \geq x\}$.
$\lfloor x\rfloor$	Abrunden, also $\lfloor x\rfloor = \max\{y \in \mathbb{Z}\,;\, y \leq x\}$.
$\exp(x)$	Exponentialfunktion von x.
$\log(x)$	Natürlicher Logarithmus von x.
$\log_b(x)$	Logarithmus von x zur Basis b.
e	Eulersche Zahl $e = \exp(1) \sim 2.71828$.
π	Kreiszahl $\pi \sim 3.14159$.
$\#\, M$	Kardinalität der Menge M.
$\deg f$	Grad des Polynoms f.
$O(f)$	Landausche O-Notation.
$n!$	Fakultät von n, also $n! = 1 \cdot 2 \cdot \ldots \cdot n$.
$\binom{n}{m}$	Binomialkoeffizient, also $\binom{n}{m} = \frac{n!}{m! \cdot (n-m)!}$.
$b \mid a$	b teilt a. 7
$b \nmid a$	b teilt nicht a. 7
$\operatorname{ggT}(a, b)$	größter gemeinsamer Teiler von a und b. 7
$\nu_p(x)$	p-adische Bewertung von x. 9
$\pi(x)$	Anzahl der Primzahlen kleiner gleich x.
$a \equiv b \pmod{n}$	a kongruent b modulo n, d.h. $n \mid (b - a)$.
$a \mod n$	Kleinster nicht-negativer Vertreter der arithmetischen Progression $a + n\mathbb{Z}$.
$\varphi(n)$	Eulersche φ-Funktion, d.h. $\varphi(n) = \#\,(\mathbb{Z}/n\mathbb{Z})^*$.
$\left(\frac{n}{m}\right)$	Legendre- bzw. Jacobisymbol.

Stichwortverzeichnis