Vorkurs für

Studierende in Mathematik und Physik

Einführung in Kryptographie
Kurzskript 2018

Felix Fontein
Institut für Mathematik
Universität Zürich
Winterthurerstrasse 190
8057 Zürich

14. September 2018

1 Einführung

In der *Kryptologie* werden zwei Themen behandelt:

- In der *Kryptographie* geht es um die Konzeption und Konstruktion von Systemen, die Informationen absichern, sei es durch Verschlüsselung oder Signierung.
- In der *Kryptoanalyse* geht es darum, Informationsabsicherungs-Systeme zu untersuchen, um ihre Sicherheit nachzuweisen oder um sie zu brechen.

In diesem Vortrag soll es um die Kryptographie gehen. Zwei Hauptthemen in der Kryptographie sind...

- ...das sichere Austauschen von Informationen, so dass sie für Unbefugte nicht lesbar sind; sowie
- ...das «Unterschreiben» (Signieren) von Nachrichten, um nachzuweisen, von wem sie stammen und dass sie nicht modifiziert worden sind.

Wir werden uns hier nur mit dem ersten Punkt beschäftigen. Nehmen wir an, dass eine Person A, nennen wir sie Alice, sicher mit Person B, nennen wir ihn Bob, kommunizieren möchte. Weiterhin gibt es eine Person E, Eve, die die Kommunikation zwischen Alice und Bob mitlesen kann. Alice möchte jedoch Bob eine Information zukommen lassen, die Eve nicht erhalten soll.

Es gibt zwei Arten von Systemen, mit denen Informationen sicher ausgetauscht werden können. Die erste Art sind sogenannte *symmetrische Verschlüsselungssysteme*, bei denen vorher ein gemeinsamer Schlüssel festgelegt werden muss. Jeder, der diesen Schlüssel hat, kann die Nachrichten mitlesen.

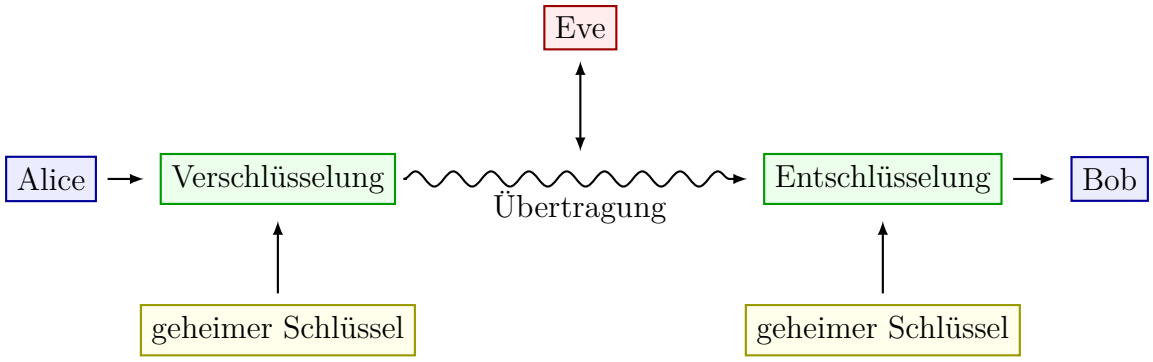

Abbildung 1: Ein symmetrisches Verschlüsselungssystem.

Solche Systeme gibt es seit vielen Jahrhunderten. Während ältere Systeme meist nicht sehr sicher sind (etwa Substitutions-Cipher oder Vigenère-Verschlüsselung), bieten moderne Systeme wie AES eine hohe Sicherheit.

Das Hauptproblem bei solchen Verfahren ist, vorher einen geheimen Schlüssel auszutauschen. Weiterhin muss jedes Paar von zwei Personen, die miteinander kommunizieren wollen, einen gemeinsamen Schlüssel haben, und dieser muss auf sicherem Weg ausgetauscht werden. Dies ist bei vielen solchen Paaren nicht gut realisierbar.

Neben symmetrischen Systemen gibt es auch asymmetrische Systeme, sogenannte *Public-Key-Systeme*. Bei einem solchen System hat Bob zwei Schlüssel: einen privaten Schlüssel, den nur er selber kennt, und einen öffentlichen Schlüssel, den jeder kennen darf, also auch Eve und Alice. Mit Hilfe des öffentlichen Schlüssels kann Alice nun eine Nachricht verschlüsseln, die nur Bob mit seinem privaten Schlüssel lesen kann. Also auch wenn Eve den öffentlichen Schlüssel sowie die verschlüsselte Nachricht kennt, kann sie diese nicht lesen, ohne den privaten Schlüssel zu kennen.

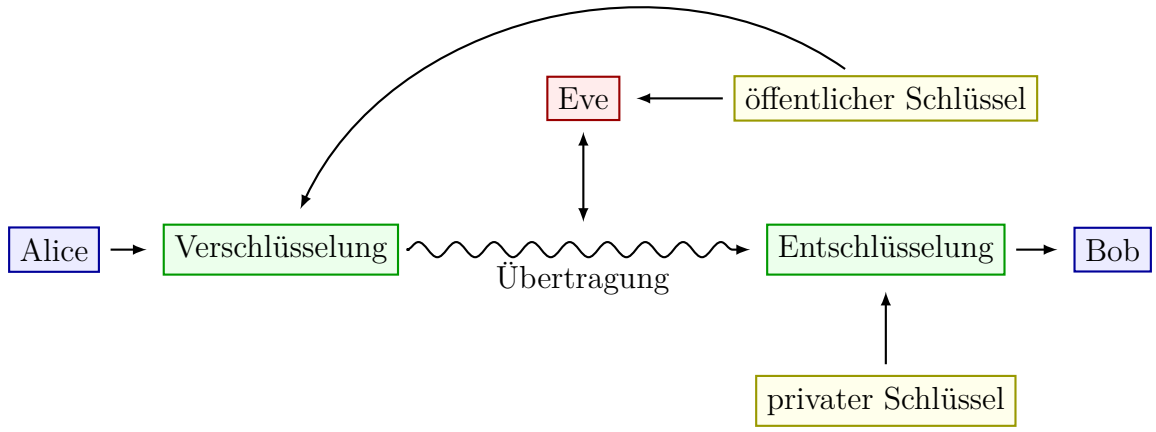

Abbildung 2: Ein asymmetrisches Verschlüsselungssystem.

Das erste praktische Beispiel für ein solches asymmetrisches Verfahren ist RSA, welches 1977 von R. Rivest, A. Shamir und L. Adleman entwickelt und veröffentlicht wurde.[1] Dieses System ist momentan das am häufigsten verwendete Public-Key-Kryptosystem; erst langsam wird es von neueren Systemen, die z. B. auf elliptischen Kurven basieren, abgelöst.

Im Gegensatz zu symmetrischen Verschlüsselungsverfahren basieren asymmetrische Verfahren ausnahmslos auf mathematischen Problemen, die meist in eine Richtung einfach sind und in eine andere Richtung schwer. RSA etwa basiert auf dem Faktorisierungsproblem: das Multiplizieren zweier sehr grosser Zahlen ist einfach, während das Zerlegen einer grossen Zahl in Primfaktoren nach momentanen Wissensstand schwer ist. Ein weiteres wichtiges Problem ist das sogenannte Diskrete-Logarithmus-Problem (DLP), welches wir gleich genauer anschauen wollen.

In diesem Kurzskript wollen wir eine dritte Methode anschauen: eine Methode, mit der Alice und Bob einen gemeinsamen Schlüssel erzeugen können. Und zwar ohne dass Eve diesen bekommt, obwohl sie die vollständige Kommunikation zwischen Alice und Bob mitlesen kann.

Die Idee basiert ebenfalls auf dem Diskreten-Logarithmus-Problem (DLP). Allgemein ist es so, dass man bei gegebenen x und y effizient x^y ausrechnen kann. Das DLP ist nun: gegeben x und $z := x^y$, finde y. Wenn man mit ganzen Zahlen arbeitet, ist $y = \log_x z = \frac{\log_{10} z}{\log_{10} x}$. Da der Zehnerlogarithmus einfach berechnet werden kann – der ganzzahlige Anteil entspricht im Wesentlichen der Anzahl der Dezimalstellen von z bzw. x – kann somit y auf einen kleinen Bereich beschränkt werden, der schnell abgesucht werden kann.

Wenn man jedoch eine Primzahl p wählt und nicht $z = x^y$, sondern z als den Rest von x^y bei Division durch p gegeben hat, ist das Problem wesentlich schwieriger, wenn man x, z und p

[1] Interessanterweise wurde erst 1997 bekannt, dass bereits 1973 C. Cocks, ein Mathematiker und Mitarbeiter des britischen GCHQ, ein äquivalentes System entwickelt hat. Dieses System wurde erst 1997 deklassifiziert und somit der Öffentlichkeit bekannt.

gegeben hat. Dies haben sich W. Diffie und M. Hellman zunutze gemacht, als sie ein Protokoll für den sicheren Schlüsselaustausch vorgestellt haben.

Dieser sogenannte *Diffie-Hellman-Schlüsselaustausch* wird ebenfalls sehr oft verwendet, etwa bei *Perfect Forward Secrecy*, welches bei https-Verbindungen immer öfter verwendet wird.

Wir wollen nun zuerst Modulo-Rechnung und schnelle Exponentiation vorstellen, und danach beschreiben wie der Diffie-Hellman-Schlüsselaustausch funktioniert.

2 Modulo-Rechnung und schnelle Exponentiation

Das wohl einfachste Beispiel zur Modulo-Rechnung ist eine (analoge) Uhr: auf dieser sind die Zahlen 1 bis 12 zu finden, und wenn man von 12 Uhr eine Stunde weitergeht, ist man wieder bei 1 Uhr.

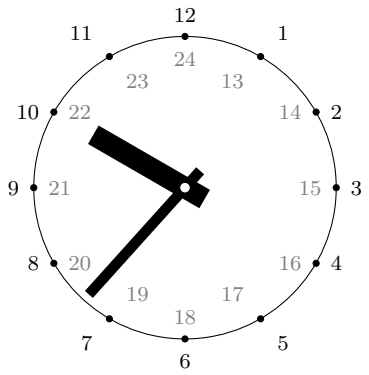

Abbildung 3: Die Uhrzeiten auf einer analogen Uhr.

Bei den Uhrzeiten identifiziert man also auf der Uhr zwei Zeiten, wenn deren Differenz ein Vielfaches von 12 ist. Man kann sich die Uhr also auch als ein unendliches Band vorstellen, auf dem alle ganzen Zahlen stehen, welches so aufgewickelt ist, dass sich zwei um ein Vielfaches von 12 unterscheidende Zahlen an der gleichen Stelle befinden.

2.1 Modulo-Rechnung

Dies kann man wie folgt verallgemeinern:

Definition 2.1. Sei $n > 1$ eine natürliche Zahl. Für zwei ganze Zahlen a, b schreiben wir $a \equiv b \pmod{n}$, wenn $b - a$ ein Vielfaches von n ist, wenn es also eine ganze Zahl m gibt mit $m \cdot n = b - a$. Wir sagen dann *«a ist kongruent zu b modulo n»*.

Dass die Uhrzeiten 9 und 21 Uhr auf der Uhr an derselben Stelle sind, kann nun durch $9 \equiv 21 \pmod{12}$ ausgedrückt werden.

Das Symbol $\equiv$ hat nicht umsonst Ähnlichkeiten mit dem Gleichheitszeichen; es verhält sich bezüglich verschiedener Operationen genauso:

Lemma 2.2. *Seien a, a', b, b' ganze Zahlen mit $a \equiv a' \pmod{n}$ und $b \equiv b' \pmod{n}$. Dann gilt auch:*

a) $a + b \equiv a' + b' \pmod{n}$;

b) $a \cdot b \equiv a' \cdot b' \pmod{n}$.

Als Anwendung wollen wir nun einfach die letzten zwei Stellen von 9551^{12345} bestimmen. Wenn m die letzten zwei Dezimalstellen vom Ergebnis sind, dann ist also $9551^{12345} = 100 \cdot q + m$, wobei q eine ganze Zahl ist. Damit ist $9551^{12345} \equiv m \pmod{100}$. Nun ist $9551 \equiv 51 \pmod{100}$, womit $9551^{12345} \equiv 51^{12345} \pmod{100}$ ist. Nun ist $51^2 = 2601 \equiv 1 \pmod{100}$, womit

$$51^{12345} = 51^{2 \cdot 6172 + 1} = (51^2)^{6172} \cdot 51 \equiv 1^{6172} \cdot 51 = 51 \pmod{100}$$

ist.

Das hier «zufällig» $51^2 \equiv 1 \pmod{100}$ ist ist kein Zufall. Meistens ist dies nicht der Fall; wären die letzten Ziffern etwa 03 gewesen, so ist erst $03^{20} \equiv 1 \pmod{100}$. Wir werden im nächsten Abschnitt eine Technik kennenlernen, wie man trotzdem effizient Exponentieren kann.

Die Relation $\equiv \pmod{n}$ ist eine *Äquivalenzrelation* auf den ganzen Zahlen $\mathbb{Z}$. Die Äquivalenzklasse von einer ganzen Zahl z ist die Menge

$$\overline{z} := \{\dots, z - 4n, z - 3n, z - 2n, z - n, z, z + n, z + 2n, z + 3n, z + 4n, \dots\}.$$

Im Beispiel $z = 9$ und $n = 12$ haben wir etwa $\overline{z} = \{\dots, -27, -15, -3, 9, 21, 33, 45, 57, 69, \dots\}$. Lemma 2.2 sagt, dass $\equiv \pmod{n}$ sogar eine *Kongruenzrelation* ist.

Definition 2.3. Wir schreiben $\mathbb{Z}_n$ für die Menge der Äquivalenzklassen $\overline{z}$ modulo n, also

$$\mathbb{Z}_n = \{\overline{z} \mid z \in \mathbb{Z}\}.$$

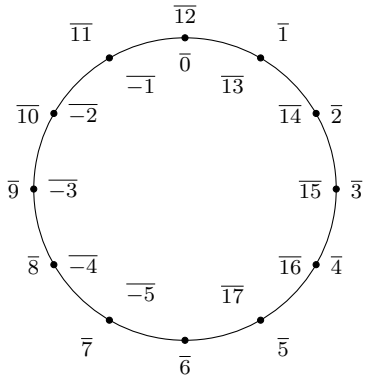

Abbildung 4: Eine Darstellung von $\mathbb{Z}_{12}$.

Auf $\mathbb{Z}_n$ können wir jetzt zwei Operationen definieren:

$$\overline{a} + \overline{b} := \overline{a + b} \qquad \text{und} \qquad \overline{a} \cdot \overline{b} := \overline{a \cdot b}.$$

Aus Lemma 2.2 folgt, dass diese Operationen *wohldefiniert* sind, also nicht von den Vertretern a und b abhängen.

In $\mathbb{Z}_n$ können wir rechnen wie in $\mathbb{Z}$; viele der aus $\mathbb{Z}$ bekannten Regeln gelten hier immer noch:

a) $+$ und $\cdot$ sind kommutativ und assoziativ;

b) $\cdot$ ist distributiv über $+$;

c) $\overline{0}$ ist das neutrale Element bzgl. $+$, d.h. $\overline{a} + \overline{0} = \overline{a}$ für alle $\overline{a} \in \mathbb{Z}_n$;

d) $\overline{1}$ ist das neutrale Element bzgl. $\cdot$, d.h. $\overline{a} \cdot \overline{1} = \overline{a}$ für alle $\overline{a} \in \mathbb{Z}_n$;

e) bezüglich $+$ gibt es inverse Elemente: zu $\overline{a} \in \mathbb{Z}_n$ hat man $\overline{a} + \overline{b} = \overline{0}$ mit $b := n - a$.

Wir bezeichnen das $\overline{b}$ aus (e) mit $-\overline{a}$, und schreiben wie gewohnt $\overline{x} - \overline{y}$ für $\overline{x} + (-\overline{y})$.

Theorem 2.4. *Zu jedem $\overline{a} \in \mathbb{Z}_n$ gibt es genau eine Zahl $b \in \{0, \ldots, n-1\}$ mit $\overline{a} = \overline{b}$. Und zwar ist b gerade der Rest bei Division von a durch n.*

Dies bedeutet, dass die Abbildung

$$\{0, \ldots, n-1\} \to \mathbb{Z}_n, \quad b \mapsto \overline{b}$$

bijektiv ist.

Wir können also $\mathbb{Z}_n$ mit der Menge $\{0, 1, \ldots, n-1\}$ identifizieren. Addition $a \oplus_n b$ und Multiplikation $a \odot_n b$ von $a, b \in \{0, 1, \ldots, n-1\}$ übertragen sich von $\mathbb{Z}_n$ auf $\{0, 1, \ldots, n-1\}$ wie folgt:

a) Es ist
$$a \oplus_n b = \begin{cases} a + b & \text{falls } a + b < n, \\ a + b - n & \text{falls } a + b \geq n. \end{cases}$$

b) Führe eine Division mit Rest aus: $a \cdot b = q \cdot n + r$, wobei q und r ganze Zahlen mit $0 \leq r < n$ sind. Dann ist $a \odot_n b = r$.

Damit ist die Menge $\{0, 1, \ldots, n-1\}$ zusammen mit den Operationen $\oplus_n$ und $\odot_n$ ein einfaches *Modell* von $(\mathbb{Z}_n, +, \cdot)$.

2.2 Schnelle Exponentiation

Möchte man 2^{13} ausrechnen, so muss man $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$ ausrechnen. Hat man nun einen Taschenrechner zur Verfügung, der zwar Multiplizieren kann, aber nicht exponentieren, so muss man zehn mal multiplizieren.

Jedoch kann man 2^{13} auch schneller ausrechnen. Es ist nämlich $2^{13} = 2^{12+1} = 2^{12} \cdot 2 = 2^{6 \cdot 2} \cdot 2 = (2^6)^2 \cdot 2 = (2^{3 \cdot 2})^2 \cdot 2 = ((2^3)^2)^2 \cdot 2 = ((2 \cdot 2 \cdot 2)^2)^2 \cdot 2$. Man kann dies mit fünf Multiplikationen ausrechnen:

$$\begin{aligned} 2 \cdot 2 &= 4 = 2^2; \\ 4 \cdot 2 &= 8 = 2^3; \\ 8^2 = 8 \cdot 8 &= 64 = (2^3)^2 = 2^6; \\ 64^2 = 64 \cdot 64 &= 4096 = (2^6)^2 = 2^{12}; \\ 4096 \cdot 2 &= 8192 = 2^{12} \cdot 2 = 2^{13}. \end{aligned}$$

Der Grund ist hier, dass $13 = 1 + 12 = 1 + 2 \cdot 2 \cdot 3 = 1 + 2 \cdot 2 \cdot (1 + 1 + 1)$ ist. Aus dieser Darstellung kann man sofort ablesen, dass man fünf Multiplikationen benötigt: jede Addition und Verdoppelung (Multiplikation mit 2) im Exponenten entspricht einer Multiplikation.

Ähnlich kann man vorgehen, wenn man 1234^{4321} ausrechnen möchte. Da

$$\begin{aligned}4321 &= 1 + 4320\\ &= 1 + 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 135\\ &= 1 + 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot (1 + 2 \cdot 67)\\ &= 1 + 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot (1 + 2 \cdot (1 + 2 \cdot 33))\\ &= 1 + 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot (1 + 2 \cdot (1 + 2 \cdot (1 + 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 1)))\end{aligned}$$

ist,[2] benötigt man 16 Multiplikationen, um damit 1234^{4321} auszurechnen. Das ist sehr wenig im Vergleich zu den 4320 Multiplikationen, die man mit der «naiven» Methode benötigt. Aus der Darstellung kann man auch direkt ablesen, wie man multiplizieren und quadrieren muss:

$$1234^{4321} = \Big(\Big(\Big(\Big(\Big(\big((((((1234)^2)^2)^2)^2)^2 \cdot 1234\big)^2 \cdot 1234\Big)^2 \cdot 1234\Big)^2\Big)^2\Big)^2\Big)^2 \cdot 1234$$

Nun ist die Zahl 1234^{4321} jedoch sehr gross: genauer gesagt hat sie 13358 Dezimalstellen! Wenn man 100 Ziffern in einer Zeile schreibt, braucht man 134 Zeilen, um das Ergebnis vollständig aufschreiben zu können.

Wenn wir allerdings nur die letzten vier Dezimalstellen wissen wollen, müssen wir modulo $10^4 = 10000$ rechnen. Das kann man mit einem normalen Taschenrechner recht schnell hin bekommen. Laut Taschenrechner ist $1234 \cdot 1234 = 1522756$, und somit $1234 \cdot 1234 \equiv 2756 \pmod{10^4}$. Mit dem Modell aus Abschnitt 2.2 hätten wir es auch als $1234 \odot_{10^4} 1234 = 2756$ schreiben können. Wieder mit dem Taschenrechner bekommt man sofort $2756 \cdot 2756 \equiv 5536 \pmod{10^4}$ heraus. Damit gilt $1234^4 = (1234 \cdot 1234)^2 \equiv 5536 \pmod{10^4}$. Wir müssen noch drei mal quadrieren: $5536^2 \equiv 7296 \pmod{10^4}$, $7296^2 \equiv 1616 \pmod{10^4}$, $1616^2 \equiv 1456 \pmod{10^4}$. Damit haben wir

$$((((1234 \cdot 1234)^2)^2)^2)^2 \equiv 1456 \pmod{10^4}.$$

Hätten wir dieses Zwischenergebnis exakt ausgerechnet, also nicht modulo 10^4, hätten wir

$$\begin{gathered}835'766'790'521'608'489'470'804'150'221'650'637'655'651'650'245'824'\\ 748'876'026'021'952'966'731'428'198'479'058'415'057'639'571'456\end{gathered}$$

herausbekommen. (Da die Zahl zu lang für eine Zeile war, ist sie auf zwei Zeilen aufgespalten.) Wie man sehen kann, haben wir die letzten vier Dezimalstellen korrekt bestimmt, ohne mit solch riesigen Zahlen zu arbeiten!

Wir haben bisher

$$1234^{4321} \equiv ((((((1456 \cdot 1234)^2 \cdot 1234)^2 \cdot 1234)^2)^2)^2)^2 \cdot 1234 \pmod{10^4}.$$

Nun ist $1456 \cdot 1234 = 1796704 \equiv 6704 \pmod{10^4}$ und somit $(1456 \cdot 1234)^2 \equiv 6704^2 \pmod{10^4}$ und wegen $6704^2 = 44943616$ haben wir

$$1234^{4321} \equiv ((((((3616 \cdot 1234)^2 \cdot 1234)^2)^2)^2)^2)^2 \cdot 1234 \pmod{10^4}.$$

Mit $3616 \cdot 1234 = 4462144$ und $2144^2 = 4596736$, und danach $6736 \cdot 1234 = 8312224$ bekommen wir

$$1234^{4321} \equiv (((((2224)^2)^2)^2)^2)^2 \cdot 1234 \pmod{10^4},$$

und $2224^2 = 4946176$, $6176^2 = 38142976$, $2976^2 = 8856576$, $6576^2 = 43243776$, $3776^2 = 14258176$ und schliesslich $8176 \cdot 1234 = 10089184$, womit $1234^{4321} \equiv 9184 \pmod{10^4}$ ist.

Wenn wir den gesamten Prozess als «Algorithmus» aufschreiben, erhalten wir folgendes:

[2] Die binäre Entwicklung von 4321 ist 1000011100001. Daraus lässt sich die Darstellung direkt ablesen: bei jeder Ziffer – ausser der ersten – findet immer eine Quadratur statt, und bei jeder auftretenden 1 – ausser der ersten – wird mit 1234 multipliziert.

(1) Setze $n := 10^4$ und $x := 1234$;
(2) Berechne $x_1 := x \odot_n x$;
(3) Berechne $x_2 := x_1 \odot_n x_1$;
(4) Berechne $x_3 := x_2 \odot_n x_2$;
(5) Berechne $x_4 := x_3 \odot_n x_3$;
(6) Berechne $x_5 := x_4 \odot_n x_4$;
(7) Berechne $x_6 := x_5 \odot_n x$;
(8) Berechne $x_7 := x_6 \odot_n x_6$;
(9) Berechne $x_8 := x_7 \odot_n x$;
(10) Berechne $x_9 := x_8 \odot_n x_8$;
(11) Berechne $x_{10} := x_9 \odot_n x$;
(12) Berechne $x_{11} := x_{10} \odot_n x_{10}$;
(13) Berechne $x_{12} := x_{11} \odot_n x_{11}$;
(14) Berechne $x_{13} := x_{12} \odot_n x_{12}$;
(15) Berechne $x_{14} := x_{13} \odot_n x_{13}$;
(16) Berechne $x_{15} := x_{14} \odot_n x_{14}$;
(17) Berechne $x_{16} := x_{15} \odot_n x$;
(18) Gebe x_{16} aus.

Wir können das ganze also mit 16 Multiplikationen von höchstens vierstelligen Zahlen und 16 Divisionen mit Rest von höchstens achtstelligen Zahlen mit 10^4 bewerkstelligen.

3 Diffie-Hellman-Schlüsselaustausch

Die Idee beim Diffie-Hellman-Schlüsselaustausch ist folgende: Alice und Bob einigen sich auf eine grosse Primzahl p sowie ein Element $g \in \{2, 3, \ldots, p-2\}$, für welches keine «zu kleine» ganze Zahl $e \geq 1$ existiert mit $g^e \equiv 1 \pmod{p}$.

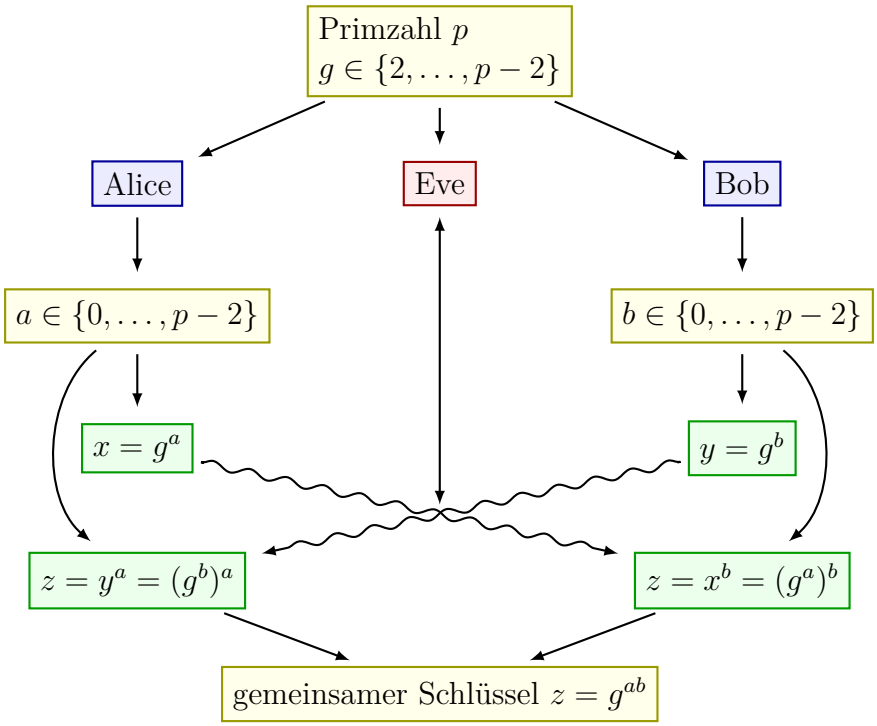

Abbildung 5: Der Diffie-Hellman-Schlüsselaustausch.

1. Dann wählt Alice zufällig eine Zahl $a \in \{0, \ldots, p-2\}$ und berechnet $x := g^a$ in $\mathbb{Z}_p$. Dies kann mit der Methode aus dem vorherigen Abschnitt effizient durchgeführt werden. Alice schickt x an Bob.

2. Bob wählt ebenfalls zufällig eine Zahl $b \in \{0, \ldots, p-2\}$ und berechnet $y := g^b$ in $\mathbb{Z}_p$. Bob schickt y an Alice.

3. Alice kann nun y^a in $\mathbb{Z}_p$ berechnen, und Bob kann x^b in $\mathbb{Z}_p$ berechnen. Das Ergebnis ist beide Male $z := g^{ab}$ in $\mathbb{Z}_p$, womit sowohl Alice wie auch Bob z einfach berechnen können.

Eve dagegen kennt zwar g, x, y und p, jedoch nicht a und b und kann somit nicht einfach z ausrechnen. Die einzige bekannte Methode, dieses Problem zu lösen, besteht darin, zuerst a' oder b' mit $g^{a'} = x$ bzw. $g^{b'} = y$ zu berechnen und dann damit $z = x^{b'} = y^{a'}$ zu berechnen. Die bisherigen Algorithmen, mit denen so ein a' bzw. b' bestimmt werden kann, sind jedoch ein Vielfaches langsamer als die Methode zum schnellen Exponentieren aus dem vorherigen Abschnitt.

4 Ausblick

Anstelle der multiplikativen Gruppe von $\mathbb{Z}_p$ verwendet man die Punktegruppen von sogenannten elliptischen Kurven für den Diffie-Hellman-Schlüsselaustausch. Beide Arten von Gruppen haben jedoch den Nachteil, dass das Diskrete-Logarithmus-Problem in ihnen von einem Quantencomputer sehr effizient gelöst werden kann – ganz im Gegensatz zu klassischen Computern, auf denen das Problem nur mit sehr viel Aufwand lösbar ist. Deswegen wird momentan nach anderen Verfahren zum Schlüsselaustausch gesucht, die sowohl für «klassische» Computer wie auch für Quantencomputer schwer zu lösen sind. Ein vielversprechendes Verfahren basiert ebenfalls auf elliptischen Kurven, verwendet jedoch nicht die Gruppenoperation, sondern Abbildungen zwischen elliptischen Kurven, sogenannte Isogenien. Weiterhin gibt es Verfahren, die auf anderen mathematischen Objekten basieren, etwa auf Gittern oder auf fehlerkorrigierenden Codes.